Design of Stream Ciphers and Cryptographic Properties of Nonlinear Functions

by

Yassir Nawaz

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2007

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.
I understand that my thesis may be made electronically available to the public.

Abstract

Block and stream ciphers are widely used to protect the privacy of digital information. A variety of attacks against block and stream ciphers exist; the most recent being the algebraic attacks. These attacks reduce the cipher to a simple algebraic system which can be solved by known algebraic techniques. These attacks have been very successful against a variety of stream ciphers and major efforts (for example eSTREAM project) are underway to design and analyze new stream ciphers. These attacks have also raised some concerns about the security of popular block ciphers. In this thesis, apart from designing new stream ciphers, we focus on analyzing popular nonlinear transformations (Boolean functions and S-boxes) used in block and stream ciphers for various cryptographic properties, in particular their resistance against algebraic attacks. The main contribution of this work is the design of two new stream ciphers and a thorough analysis of the algebraic immunity of Boolean functions and S-boxes based on power mappings.

First we present WG, a family of new stream ciphers designed to obtain a keystream with guaranteed randomness properties. We show how to obtain a mathematical description of a WG stream cipher for the desired randomness properties and security level, and then how to translate this description into a practical hardware design. Next we describe the design of a new RC4-like stream cipher suitable for high speed software applications. The design is compared with original RC4 stream cipher for both security and speed.

The second part of this thesis closely examines the algebraic immunity of Boolean functions and S-boxes based on power mappings. We derive meaningful upper bounds on the algebraic immunity of cryptographically significant Boolean power functions and show that for large input sizes these functions have very low algebraic immunity. To analyze the algebraic immunity of S-boxes based on power mappings, we focus on calculating the bi-affine and quadratic equations they satisfy. We present two very efficient algorithms for this purpose and give new S-box constructions that guarantee zero bi-affine and quadratic equations. We also examine these S-boxes for their resistance against linear and differential attacks and provide

a list of S-boxes based on power mappings that offer high resistance against linear, differential, and algebraic attacks. Finally we investigate the algebraic structure of S-boxes used in AES and DES by deriving their equivalent algebraic descriptions.

Acknowledgements

I would like to thank my Ph.D. supervisor, Dr. Guang Gong for her guidance and support during this work. I have always found her to be very caring and encouraging. I would also like to thank Dr. Alfred Menezes, Dr. Anwar Hasan, Dr. Mark Aagard, and Dr. Andrew Klapper for their suggestions and insightful comments that have helped immensely in improving the quality of this thesis.

This research was largely supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Post Graduate Doctoral Scholarship. I would like to thank NSERC for this generous support. I would also like to thank my fellow researchers at the Communications Security Lab, especially, Kishan Gupta, Nam Yul Yu, Katrin Hoeper, and Xinxin Fan. I have always benefited from having academic discussions with them.

My family has also played a significant role in my success. I would like to thank my wife, Shahla for her understanding and support. Most of all I would like to thank my parents, especially my father, for inspiring and actively encouraging me to pursue a doctoral degree. I therefore dedicate this thesis to my father, Dr. Muhammad Nawaz Chaudhry.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Cryptography is the science of providing secure services. Until 1970s cryptography was considered the domain of military and governments only. However the ubiquitous use of computers and the advent of internet has made it an integral part of our daily lives. Today cryptography is at the heart of many secure applications such as online banking, online shopping, online government services such as filing personal income taxes, cellular phones, and wireless LANS (Local Area Networks) etc. In this chapter we provide an introduction to some cryptographic primitives which are used to design secure applications.

1.1 Requirements for Cryptography

Cryptography is generally used in practice to provide four services: *privacy, authentication, data integrity* and *non-repudiation*. The goal of privacy is to ensure that communication between two parties remain secret. This often means that the contents of the communication are secret, however in certain situations the very fact that communication took place must be a secret as well. Encryption is generally used to provide privacy in modern communication. Authentication of one or both parties during a communication is required to ensure that information is being exchanged with the legitimate party. Passwords are common examples of one-way authentication in which users authenticate themselves to gain access to a system.

The modern authentication procedures are much more complex where individual devices may need to authenticate themselves to previously unknown systems or devices for communication. The goal of data integrity is to ensure that the contents of a communication between two parties have not been altered by an attacker. Hash functions and message authentication codes are generally used to ensure integrity of a message. Non-repudiation is used to prevent a party from denying the existence of a communication or a contract in future.

1.2 Cryptographic Primitives

A cryptographic primitive is a specific tool used to provide a cryptographic service. Cryptographic primitives are used according to a well designed set of protocols to ensure the overall system security. We can divide the cryptographic primitives in two categories: symmetric key primitives and asymmetric key primitives. The encryption schemes used for providing privacy are similarly divided into symmetric key encryption and asymmetric key encryption or public key encryption.

1.2.1 Symmetric Key Encryption

In symmetric key encryption a secret key is shared between the sender and the receiver. The word "symmetric" refers to the fact that both sender and receiver use the same key to encrypt and decrypt the information. The secret key must be shared over a secure channel before the communication. Two types of cryptographic primitives are used for symmetric key encryption: block ciphers and stream ciphers.

Block Ciphers: A block cipher is a symmetric key cryptographic primitive which takes as input an n-bit block of plaintext and a secret key and outputs an n-bit block of ciphertext using a fixed transformation. Figure 1.1 shows the general structure of a block cipher. The common block sizes are 64 bits, 128 bits and 256 bits. For a fixed key the block cipher defines a permutation on the n-bit input. The building blocks of modern block ciphers are substitutions and permutations. Substitutions are generally performed to provide nonlinearity in the output and

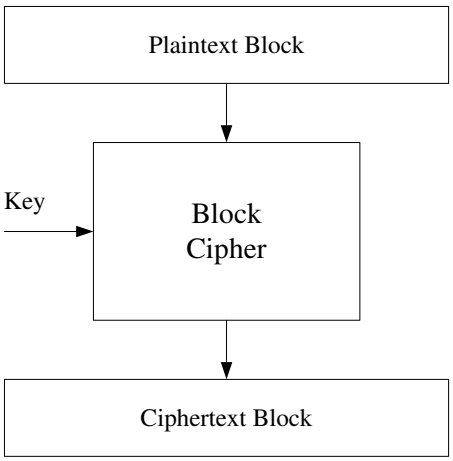

Figure 1.1: General structure of a block cipher

permutations are used to diffuse the input and the key. These substitution and permutation operations in a cipher are repeated several times to obtain a highly nonlinear transformation of the input bits.

The two most famous block ciphers are DES (Data Encryption Standard) [31] and AES(Advanced Encryption Standard) [26]. DES was designed in 1970 and has been extensively used in the past three decades. It has a block size of 64 bits and a secret key of 56 bits. AES was standardized in 2002 to replace DES since 56 bit key is too small for the computing power of todays computers. AES has an increased key length of 128, 192, or 256 bits and a block size of 128 bits.

Stream Ciphers: Unlike block ciphers, which encrypt large blocks of plaintext using a fixed transformation, stream ciphers encrypt individual digits of plaintext using a time-varying transformation. Figure 1.2 shows the general structure of a stream cipher. The size of the digit can vary depending on the design constraints, application and the underlying platform. A stream cipher usually consists of a pseudo random generator. The generator takes as input a secret key and then generates a pseudo random sequence of digits known as the running keystream. The keystream digits are XORed with the plaintext digits to obtain ciphertext digits.

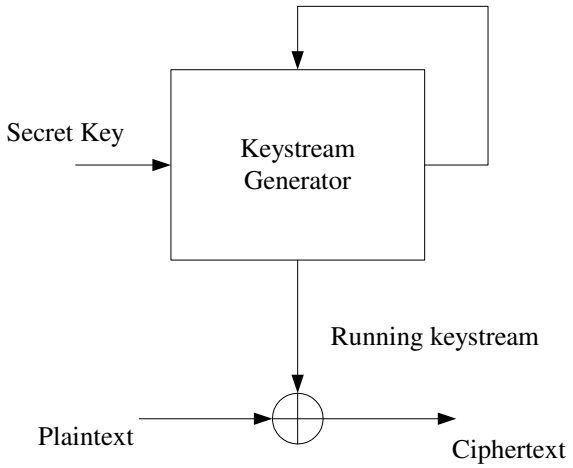

Figure 1.2: General structure of a stream cipher

1.2.2 Public Key Encryption

Public or asymmetric key encryption uses two different keys for encryption and decryption. The key used for the encryption is known as the public key since everyone has access to it. Anyone can encrypt a message using this key, however the encrypted message can only be decrypted using the private key that corresponds to the public key used in encryption. Therefore only the person in possession of private key can decrypt the message. The encryption process is therefore considered to be one-way and hard to invert unless a trap door is used. This trap door is provided by the private key. Public key encryption is very flexible however it requires the existence of a trusted third party that can authenticate the public keys. Moreover public key encryption schemes are very slow when compared to symmetric key schemes and can not be used for bulk encryption. Most practical systems use a combination of public and private key encryption. A public key scheme is first used for the key agreement between two parties which then switch to faster symmetric key encryption.

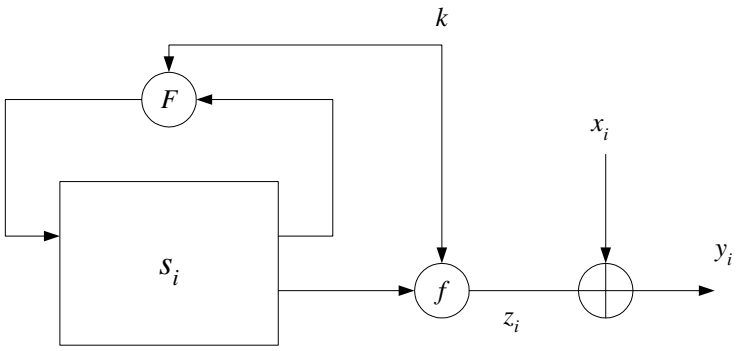

Figure 1.3: Synchronous stream cipher

1.3 Design of Stream Ciphers

Stream ciphers are commonly subdivided into two categories: synchronous and self-synchronizing stream ciphers. The theory of finite state machines can be used to describe both categories of stream ciphers [109].

1.3.1 Synchronous Stream Ciphers

In synchronous stream ciphers the keystream is generated from the key independently of the plaintext and the ciphertext. Figure 1.3 shows the general structure of a synchronous stream cipher. Let x_i, y_i, z_i and s_i be the plaintext digit, ciphertext digit, keystream digit and the internal state of the cipher at time i. Then the encryption process can be described by the following equations

$$s_{i+1} = F(k, s_i) \tag{1.1}$$

$$z_i = f(k, s_i) \tag{1.2}$$

$$y_i = x_i + z_i \tag{1.3}$$

where F is the next-state function and f is the output function. The ciphertext is obtained by the bitwise addition of the keystream digit and the plaintext digit. The ciphertext at the receiver is decrypted by producing the same keystream and adding it to the ciphertext. The correct decryption requires the sender and the

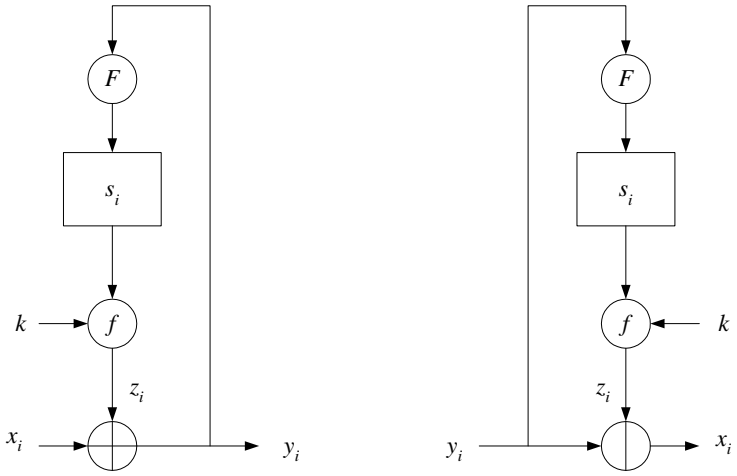

Figure 1.4: Self-synchronous stream cipher

receiver to be perfectly synchronized. A loss of synchronization means all plaintext digits decrypted after the loss will be in error.

1.3.2 Self-Synchronizing Stream Ciphers

In self-synchronizing stream ciphers the keystream is generated from the key as well as a fixed number of previous ciphertext digits. Figure 1.4 shows the general structure of a self-synchronizing stream cipher. The encryption by a self-synchronous stream cipher can be described as

$$s_i = F(y_{i-1}, y_{i-2}, ..., y_{i-v}) \quad (1.4)$$

$$z_i = f(k, s_i) \quad (1.5)$$

$$y_i = x_i + z_i \quad (1.6)$$

The keystream produced at time i depends on v previous ciphertext symbols. Therefore if there is a transmission error the cipher will resynchronize after v correct symbols.

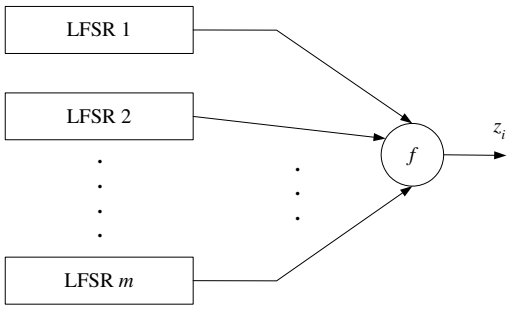

Figure 1.5: Combination generator

1.3.3 Classical Stream Cipher Designs

Most practical stream ciphers are based on linear feedback shift registers (LFSRs). An LFSR of length n is an n stage register with a linear feedback function. During its operation contents of each storage unit are shifted to the next unit and the output of the feedback function is fed to the last storage unit. If the feedback function of the LFSR is primitive and its initial state is a non-zero state, then the output sequence produced by the LFSR has the maximum period of $2^n - 1$. This sequence is known as the maximum-length sequence or simply m-sequence. The m-sequences have very good randomness properties such as maximum period, balance, ideal n-tuple distribution, ideal run distribution and ideal autocorrelation. These properties make them ideal for use in stream ciphers. However m-sequences have one drawback: their low linear complexity where linear complexity refers to the length of the shortest LFSR required to generate that sequence. Berlekamp-Massey algorithm provides an efficient way of computing the linear complexity of a sequence. For an m-sequence of length $2^n - 1$ this algorithm only requires $2n$ consecutive symbols to determine the feedback polynomial of the LFSR and hence the entire sequence. Therefore in practical stream cipher designs a large linear complexity of the keystream is obtained by a nonlinear transformation of the LFSR output sequence. These methods of transformation define three general design categories of stream ciphers: combination generators, filter generators and clock controlled generators. The three types of generators are shown in Figures 1.5, 1.6, and 1.7 respectively. In combination generator the output of several LFSRs is

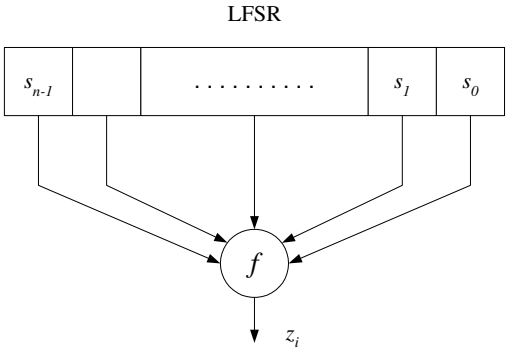

Figure 1.6: Filter generator

combined by a Boolean function f to produce the keystream. To produce a random and secure keystream the Boolean function must provide high nonlinearity and satisfy certain criteria. We will discuss these criteria in more detail in Section 1.5. Filter generators only use a single LFSR. The keystream is generated by filtering the contents of the LFSR by a Boolean function. The design of filter and combination generators using pure Boolean functions involves certain tradeoffs. Some designs avoid these tradeoffs by using filters and combiners with memory. Two famous examples are E_0 [12] and SNOW [37].

The clock controlled generators destroy the linearity of the LFSR output by using irregular clocking. Two popular clock controlled generators are the stop-and-go generator and shrinking generator. In stop-and-go generator two LFSRs are used. One is clocked regularly and its output is used to clock the second LFSR. The output of second LFSR is the output of the generator. If the output of the first LFSR is 1, second LFSR is clocked to produce a new symbol otherwise the previous symbol is repeated. The shrinking generator also uses 2 LFSRs. Instead of controlling the clocking of the second LFSR, one LFSR is used to shrink the output of the second LFSR. If the output of the first LFSR is 1 second LFSR produces an output symbol, otherwise the output symbol is discarded. Some stream cipher designs combine irregular clocking with a filter or a combination generator for example LILI 128 [16, 30].

Although most stream ciphers are based on LFSRs, there are some stream

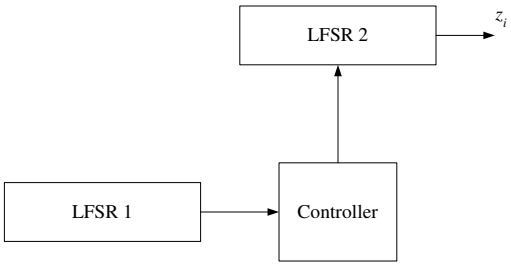

Figure 1.7: Clock controlled generator

ciphers which do not use LFSR as a component. RC4 is a very famous stream cipher which falls into this category. The internal state of the cipher consists of a permutation of 256 elements, numbered from 0 to 255, which changes with time. The keystream is generated by selecting an element as an output from the state randomly. Most of these stream ciphers are word oriented i.e. output several bits in each cycle and have been specifically designed for software applications.

1.3.4 Standardization of Stream Ciphers

Stream ciphers have been used in many information processing applications. The three significant ones are A5/1 [16] in GSM networks, E_0 [12] in bluetooth standard and RC4 [106] in WEP (802.11 wireless LAN standard). However practical attacks have been discovered on all three encryption schemes [73, 110, 49].

Two major standardization efforts that included stream ciphers were the NESSIE (New European Schemes for Signatures, Integrity and Encryption) project [89] and the International Standards Organization's ISO/IEC 18033 standard. Various cryptographic primitives including a block cipher were standardized in the NESSIE project. However all the stream cipher proposals submitted to NESSIE were rejected due to the discovery of cryptanalytic attacks. The ISO/IEC 18033 standard on the other hand selected two stream ciphers: SNOW 2.0 [38] and MUGI [114].

In 2005 a new project, eSTREAM, was launched by ECRYPT (European Network of Excellence for Cryptology) [36]. This is by far the largest effort ever undertaken to design and identify secure and efficient stream ciphers. Researchers were

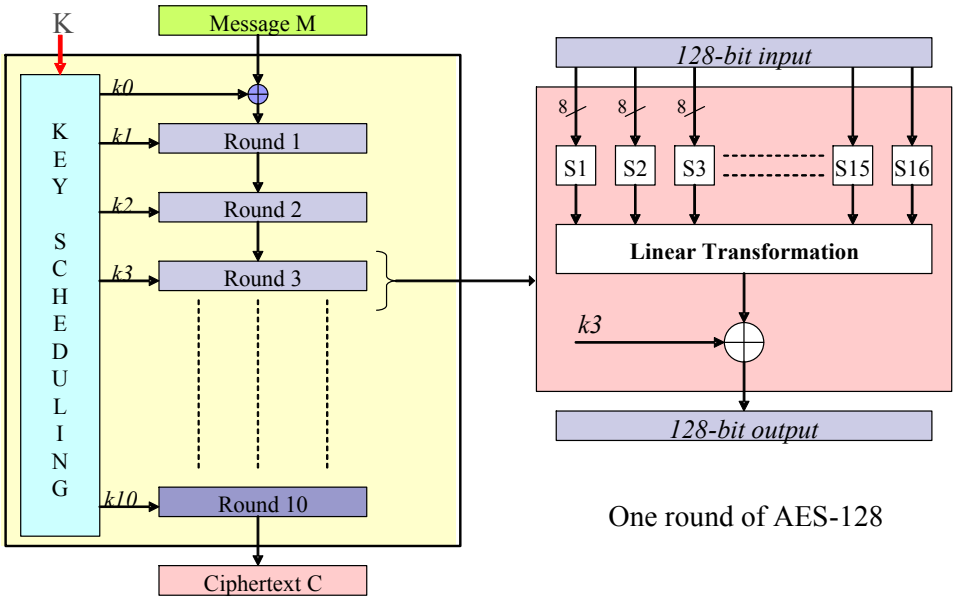

Figure 1.8: Advanced Encryption Standard (AES)-128

invited to submit stream cipher proposals in two categories: High performance software applications and hardware applications with restricted resources. These submissions have been subjected to rigorous cryptanalysis and have resulted in the enhancement of our understanding of stream cipher design. The final report of the project is expected in January 2008. The WG stream cipher presented in Chapter 3 is one of the submissions to eSTREAM project.

1.4 Design of Block Ciphers

The encryption process in block ciphers consists of transforming the data in several rounds of operations. Each round consists of two major operations: nonlinear transformation to add confusion and linear transformation to add diffusion to the input bits. In addition to this block ciphers have a key scheduling algorithm which expands a secret key into several round keys. Figure 1.8 shows the high level design of block cipher AES-128. The cipher has a key scheduling algorithm that takes 128-bit secret key as an input and expands it into eleven 128-bit round keys. The

encryption part of the algorithm consists of ten rounds. All rounds are identical except the last round which is slightly different. For more details on the design of AES please refer to [26]. The nonlinear transformation in each round consists of several substitution boxes (or S-boxes). The 128-bit input to each round is first divided into 16 byte sized chunks each of which is then transformed by a nonlinear S-box. This is followed by a linear transformation of the entire 128-bit block followed by the addition of the round key. The S-boxes constitute the only nonlinear transformation in the AES-128. This is true for most block ciphers.

1.5 Nonlinear Transformations in Cipher Design

Nonlinear transformations are at the heart of symmetric cipher design. The nonlinear transformations usually consist of Boolean functions and S-boxes. In some ciphers simple processor operations are used to create a highly nonlinear transformation.

Stream Ciphers: Most shift register based stream ciphers use nonlinear Boolean functions to filter the shift register's output. Some word oriented design also use S-boxes, i.e., vector Boolean functions. Others use popular bitwise operations (such as shift, rotate, AND and XOR etc) found on modern processors to obtain a nonlinear transformation.

Block Ciphers: Almost all block ciphers use S-boxes to introduce nonlinearity in the input bits. In some designs S-boxes with compact algebraic representations are used whereas in other designs they are chosen randomly according to a fixed criteria.

Boolean Functions: The Boolean functions used in stream ciphers must satisfy certain properties to ensure resistance against various attacks. A Boolean function must have high algebraic degree, high nonlinearity and good correlation immunity to prevent various types of correlation attacks. It must also have high algebraic immunity to provide resistance against the algebraic attacks.

S-Boxes: S-boxes are primarily used in block ciphers and are required to provide resistance against linear and differential cryptanalysis. As a result of the algebraic attacks it is desirable to have S-boxes which do not satisfy very low degree multivariate equations involving input and output bits.

1.6 Algebraic Attacks

The concept of algebraic attacks is not new however it has only been used recently to attack practical ciphers. The idea is to use the plaintext, ciphertext pairs and the structure of the cipher to derive a system of linear or nonlinear equations in terms of the unknown key. The system can then be solved to recover the key. The feasibility of the attack depends on the size and nature of the system of equations. Several stream ciphers with high resistance to correlation and other attacks were successfully attacked in [20,21,75,5]. However these attacks have been less successful against popular block ciphers and are somewhat controversial. Still these attacks have demonstrated that most of the popular block ciphers can be easily reduced to a system of low degree multivariate equations. The difficulty of solving such system is not very clear at the moment. Complexity or success of algebraic attacks against a block or stream cipher depends heavily on the nonlinear transformation(s) used in that cipher.

1.6.1 Boolean Functions and Algebraic Attacks

The success of the algebraic attacks against many LFSR based stream ciphers is due to the discovery by Courtois et. al, that the Boolean functions used in these ciphers have an undesirable property: the algebraic degree of these functions can be reduced by multiplying them with appropriate low degree functions. In order to understand the complete attack we consider a nonlinear filter generator shown in Figure 1.9 with a binary LFSR of length n and Boolean function of algebraic degree d. The generator is loaded with an n bit key, k, and it produces a keystream, z. We assume that the attacker has access to a segment of keystream z and his goal is to recover the secret key k that was loaded in the LFSR. Since the feedback polynomial

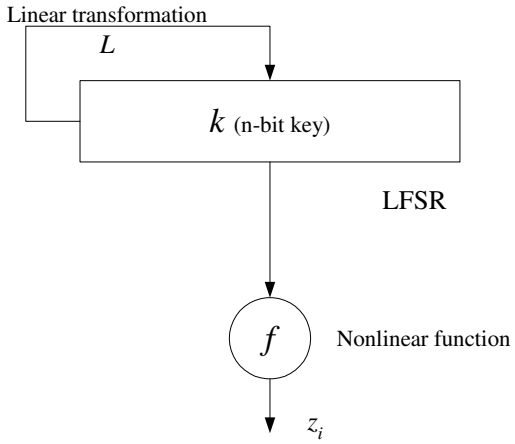

Figure 1.9: Algebraic attack on filter generator

of the LFSR is linear the state update function consists of a linear transformation L. The contents of the LFSR are filtered by a nonlinear function f to produce the keystream. Since L, z, and f are known, for each keystream digit z_i we can write a nonlinear equation, i.e.,

$$\begin{aligned} f(k) + z_0 &= 0 \\ f(L(k)) + z_1 &= 0 \\ f(L^2(k)) + z_2 &= 0 \\ \cdots &\quad \cdots \end{aligned} \tag{1.7}$$

The only unknown in the above equations is the secret key k. However this is a system of nonlinear equations and there are no efficient techniques for solving such systems. One solution is to first linearize the system by replacing all the monomial terms with degree 2 or higher with new variables. This will result in a system of linear equations which can easily be solved by algorithms such as Gaussian Elimination. To obtain a solvable system, there must be as many equations as the number of unknown variables. The number of monomial terms in n variables of degree $\leq d$ and therefore the number of distinct variables after linearizing the system is $\sum_{i=0}^{d} \binom{n}{i}$. This is the complexity of the attack and for a fixed n depends on the algebraic degree d of the Boolean function.

Courtois showed in [20] that this complexity can be reduced if we can find low degree multiple of function f, i.e., if we can find a Boolean function g such that product $f \times g$ has degree less than $deg(f)$, where $deg(f)$ is the degree of function f. Consider the nonlinear equation $f(L^i(k)) = 0$; the overall degree of the equation is $deg(f)$. If we multiply this equation by g we get $f(L^i(k)) \times g = 0$. Since the degree of $f \times g < deg(f)$, the overall degree of the new equation is lower than $deg(f)$. Therefore the newer equation has fewer monomial terms and the complexity of solving the linearized system is reduced.

1.6.2 S-Boxes and Algebraic Attacks

The algebraic attacks against block ciphers also come from the observation that the S-boxes used in most block ciphers have an interesting algebraic property: they can be expressed as an overdefined system of low degree multivariate algebraic equations. Therefore by expressing all the S-boxes in a block cipher by low degree multivariate equations, and expressing the linear transformations as linear multivariate equations we can reduce the entire block cipher to a system of low degree algebraic equations. The solution of this system recovers the secret key of the cipher. For example consider block cipher AES-128 as shown in Figure 1.8. The encryption algorithm consists of 160 identical S-boxes. Each S-box has 8 binary inputs and 8 binary outputs and can be expressed by 23 multivariate quadratic equations. If each binary input and output of an S-box is represented as a binary variable, we can obtain $16 \times 23 = 3680$ quadratic equations in $160 \times 16 = 2560$ variables. Similarly there are eleven 128 bit to 128 bit linear transformation layers in AES which give us $11 \times 128 = 1408$ linear equations. Now we have an overdefined system of quadratic equations the solution of which recovers the key. However the exact complexity of solving such systems of algebraic equations is not known. There are several algorithms that have been shown to solve small systems of quadratic equations but they are based on heuristics and it is not known how well they scale to large systems [25, 39, 40]. Still the property that most block ciphers can easily be reduced to a simple but large system of quadratic equations is undesirable. We will address this issue in Chapter 6.

1.7 Thesis Overview

This thesis is about the design of symmetric ciphers, in particular stream ciphers, and the analysis of the cryptographic properties of the nonlinear transformations used in these ciphers. The thesis is organized as follows.

We begin the thesis with an introduction to symmetric cryptography and brief review of the combinatorial objects used in the design of block and stream ciphers. In Chapter 2 we present the definitions that will be used in this thesis. In Chapter 3 we present WG, a family of new stream ciphers designed to obtain a keystream with guaranteed randomness properties. We review the cryptographic properties of the WG sequences and investigate the feasibility of constructing a practical stream cipher based on these sequences. We show how to obtain a mathematical description of a WG stream cipher for the desired randomness properties and security level, and then how to translate this description into a practical hardware design. We analyze the security of our design against various attacks and list the cryptanalysis that has been performed on WG so far.

In Chapter 4 we describe the design of a new RC4-like stream cipher suitable for high speed software applications. The attacks on the original RC4 stream ciphers are reviewed and its performance limitations on modern 32/64 bit processors is examined. This is followed by a new RC4-like stream cipher proposal which attempts to exploit the 32/64 bit architecture of the modern processors. Finally we present the distinguishing attacks and countermeasures proposed on our design so far.

The second part of this thesis closely examines the algebraic immunity of Boolean functions and S-boxes based on power mappings. We begin Chapter 5 by developing techniques to analyze Boolean functions based on power mappings in finite fields. We then use these techniques to derive meaningful upper bounds on the algebraic immunity of cryptographically significant functions, i.e., functions based on inverse, Kasami, Niho, and Dobbertin exponents. We show that for large input sizes these functions have very low algebraic immunity. Finally these results are generalized, first to monomial trace functions, and then to all polynomial functions.

In Chapter 6 we turn our attention to the algebraic immunity of S-boxes based

on power mappings as they are widely used in modern block ciphers such as AES. We first develop efficient techniques to count the number of bi-affine and quadratic equations satisfied by S-boxes based on power mappings. From these techniques we develop two algorithms and analyze various classes of S-boxes used in the design of block and stream ciphers. This is followed by the construction of S-boxes that satisfy zero or very small number of quadratic equations. Finally we examine these S-boxes for their resistance against linear and differential attacks and provide a list of S-boxes based on power mappings that offer high resistance against linear, differential, and algebraic attacks.

In Chapter 7 we study algebraic structure of S-boxes used in AES and DES by deriving their equivalent algebraic descriptions. We use these descriptions to determine the number of monomials in their polynomial and trace representations, and linear complexity of the sequences that correspond to their component functions. We conclude this thesis in Chapter 8 by summarizing the significance of our work. We also propose how to extend this work in future.

Chapter 2

Definitions and Preliminaries

In this chapter we provide definitions and necessary preliminary material that will be used in the thesis. Most of the definitions and concepts provided in this chapter can be found in [52, 67, 65].

2.1 Finite Fields

- $\mathbb{F}_2 = GF(2)$ is the finite field with 2 elements: 0 and 1.
- $\mathbb{F}_{2^n} = GF(2^n)$ is the extension field of $GF(2)$ with 2^n elements.
- $\mathbb{F}_2^n$ is the vector space over $\mathbb{F}_2$ with a set of all binary n-tuples.
- $\mathbb{Z}_n$ is the ring of integers modulo n.

Trace Function: For positive integers n and m, let $m \mid n$, then the trace function, $Tr_m^n(x)$, from $\mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^m}$ is defined as:

$$Tr_m^n(x) = \sum_{i=0}^{n/m-1} x^{2^{mi}}, x \in \mathbb{F}_{2^n}.$$

Polynomial Basis: Consider the finite field $\mathbb{F}_{2^m}$ and let α be a root of the primitive polynomial that generates $\mathbb{F}_{2^m}$. Then $\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\}$ is a polynomial basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$.

Normal Basis: Consider the finite field $\mathbb{F}_{2^m}$ and let γ be an element of $\mathbb{F}_{2^m}$ such that $\{\gamma, \gamma^{2^1}, \gamma^{2^2}, \cdots, \gamma^{2^{m-1}}\}$ is a basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. Then $\{\gamma, \gamma^{2^1}, \gamma^{2^2}, \cdots, \gamma^{2^{m-1}}\}$ is a normal basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$.

Cyclotomic Coset: A cyclotomic coset C_s modulo $2^n - 1$ is defined as

$$C_s = \{s, s \cdot 2, \cdots, s \cdot 2^{n_s - 1}\},$$

where n_s is the smallest positive integer such that $s \equiv s2^{n_s} \bmod (2^n - 1)$. The subscript s is chosen as the smallest integer in C_s, and s is called the coset leader of C_s. The computations of cosets are performed in Z_{2^n-1}, the residue integer ring modulo $(2^n - 1)$. For example the cyclotomic cosets modulo 15 are: $C_0 = \{0\}, C_1 = \{1, 2, 4, 8\}, C_3 = \{3, 6, 12, 9\}, C_5 = \{5, 10\}, C_7 = \{7, 14, 13, 11\}$, where {0,1,3,5,7} are coset leaders modulo 15.

Hamming Weight: The Hamming weight of an integer i is the number of nonzero coefficients in the binary representation of i and is denoted by $\mathrm{H}(i)$.

2.2 Boolean Functions

We consider the domain of an n-variable Boolean function to be the vector space $(\mathbb{F}_2^n, +)$ over $\mathbb{F}_2$, where $+$ is used to denote the addition operator over both $\mathbb{F}_2$ and the vector space $\mathbb{F}_2^n = \{x_1, x_2, \cdots, x_n | x_i \in \mathbb{F}_2\}$ and n is a positive integer.

Algebraic Normal Form: Any Boolean function has a unique representation as a multivariate polynomial over $\mathbb{F}_2$, called the algebraic normal form (ANF),

$$h(x_1, \ldots, x_n) = a_0 + \sum_{1 \le i \le n} a_i x_i + \sum_{1 \le i < j \le n} a_{i,j} x_i x_j + \ldots + a_{1,2,\ldots,n} x_1 x_2 \ldots x_n,$$

where the coefficients $a_0, a_i, a_{i,j}, \ldots, a_{1,2,\ldots,n} \in \mathbb{F}_2$.

Algebraic Degree: The algebraic degree, $\deg(h)$, of a Boolean function h is the number of variables in the highest order term with non zero coefficient.

Affine Function: An n variable function l is called affine if it is of the form $l(x_1, \ldots, x_n) = a_0 + \sum_{1 \leq i \leq n} a_i x_i$ where the coefficients $a_0, a_i \in \mathbb{F}_2$. If $a_0 = 0$, the function is called linear.

Walsh Transform: The Walsh transform (WT) of an n-variable Boolean function h is an integer valued function $W_h : \{0,1\}^n \rightarrow [-2^n, 2^n]$ defined by (see [67, page 414])

$$W_h(u) \quad = \quad \sum_{w \in \mathbb{F}_2^n} (-1)^{h(w) \oplus \langle u, w \rangle}, u \in \mathbb{F}_2^n. \tag{2.1}$$

$\{W_h(u) | u \in \mathbb{F}_2^n\}$ is called the *spectrum* of h. Note that Walsh transform of $h(x)$ is actually the Fourier transform of $(-1)^{h(x)}$.

Nonlinearity: Nonlinearity of a Boolean function h measures the distance of the Boolean function from the set of all affine functions. The nonlinearity $nl(h)$ of an n-variable Boolean function h, can be written as

$$nl(h) = 2^{n-1} - \frac{1}{2} \max_{u \in \mathbb{F}_2^n} |W_h(u)|.$$

Boolean functions used in stream ciphers must have high nonlinearity. A high nonlinearity weakens the correlation between the input and output and prevents the attacker from using using linear approximations of the function.

Correlation Immunity: Let $X_1, X_2, ..., X_n$ be independent binary random variables with equal probability of 0 and 1. A Boolean function $h(x_1, x_2, ..., x_n)$ is said to be t-th order correlation immune, if for each subset of t variables $X_{i_1}, X_{i_2}, ..., X_{i_t}$ with $1 \leq i_1 \leq i_2 \cdots \leq i_t \leq n$, the random variable $Z = h(X_1, X_2, ..., X_n)$ is statistically independent of the random vector $(X_{i_1}, X_{i_2}, ..., X_{i_t})$. A Boolean function

which is t-th order correlation immune and is also balanced is called t-th resilient. The Boolean function used in a nonlinear combiner must have high correlation immunity. If the combiner function is not correlation immune then the attacker can find correlations between the keystream and the contents of one of the LFSRs. This allows the attacker to mount a divide and conquer attack in which internal state of each LFSR is recovered independent of the other LFSRs.

2.3 Polynomial Functions:

Any non-zero polynomial function f: $\mathbb{F}_{2^n} \rightarrow \mathbb{F}_2$, can be represented as a sum of trace functions [52, page 178]:

$$f(x) = \sum_{k \in \Gamma(n)} Tr_1^{n_k}(A_k x^k) + A_{2^n-1} x^{2^n-1}, A_k \in \mathbb{F}_{2^{n_k}}, A_{2^n-1} \in \mathbb{F}_2,$$

where $\Gamma(n)$ is the set consisting of all coset leaders modulo $2^n - 1, n_k$ is the size of the coset C_k, and $Tr_1^{n_k}(x)$ is the trace function from $\mathbb{F}_{2^{n_k}} \rightarrow \mathbb{F}_2$. If $f(x)$ is balanced, we have [52]

$$f(x) = \sum_{k \in \Gamma(n)} Tr_1^{n_k}(A_k x^k), A_k \in \mathbb{F}_{2^{n_k}}, x \in \mathbb{F}_{2^n}. \tag{2.2}$$

Algebraic Degree: The algebraic degree of a polynomial function, f, denoted by $deg(f)$, is given by the largest w such that $A_k \neq 0$ and $H(k) = w$.

Correspondence Between Boolean and Polynomial Functions: There is a natural correspondence between Boolean functions h and polynomial functions f [52, page 334]. Let $\{\alpha_0, \ldots, \alpha_{n-1}\}$ be a basis for $\mathbb{F}_{2^n}$, then this correspondence is given by:

$$h(x_0, \ldots, x_{n-1}) = f(x), x = \sum_{i=0}^{n-1} x_i \alpha_i, x_i \in \mathbb{F}_2.$$

Monomial Function: A monomial or single trace term function is a function from $\mathbb{F}_{2^n} \rightarrow \mathbb{F}_2$ that can be represented by a single trace term, $f(x) = Tr_1^n(\beta x^t)$ where

$\beta \in \mathbb{F}_{2^n}$ and t is the coset leader of C_t.

Discrete Fourier Transform [52] : For f: $\mathbb{F}_{2^n} \rightarrow \mathbb{F}_2$, the Discrete Fourier Transform (DFT) of f is defined as

$$A_k = \sum_{x \in \mathbb{F}_{2^n}^*} [f(x) + f(0)]x^{-k}, k = 0, 1, \ldots, 2^n - 1.$$

The sequence $(\underline{A})$=A_k is also called the spectral sequence. The inverse DFT formula is given as :

$$f(x) = \sum_{k=0}^{2^n-1} A_k x^k, x \in \mathbb{F}_{2^n}^*.$$

Therefore given the output of the function f we can compute its polynomial representation.

2.4 S-Boxes

An (n, m) S-box (or vector Boolean function) is a map F: $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ and has component Boolean functions $f_1, \cdots, f_m$.

Algebraic Degree: We define the degree of an (n, m) S-box, F, to be the minimum of the degrees of all non zero linear combinations of its component functions.

Nonlinearity: Nonlinearity of an (n, m) S-box is defined as the minimum nonlinearity of all the non zero linear combinations of its component functions.
An S-box used in a block cipher must have high nonlinearity to prevent linear cryptanalysis. In linear cryptanalysis an attacker exploits the probabilistic linear relations between the input and output of an S-box. If the nonlinearity of an S-box is low then these relations hold with high probability which leads to linear approximation of the cipher that is used to recover the secret key. On the other hand an S-box with high nonlinearity weakens these probabilistic relations and make linear cryptanalysis very hard.

Maximally Nonlinear: Let $\mathbb{F}_{2^n}$ be the finite field with 2^n elements. Consider a mapping F: $\mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n}$. If n is odd F is called maximally nonlinear [34] if the Walsh spectra of all nonzero linear combinations of its component functions have precisely 3 values $\{0, \pm 2^{\frac{n+1}{2}}\}$. If n is even then it is conjectured [34] that maximum achievable nonlinearity by F is $2^{n-1} - 2^{\frac{n}{2}}$. If F achieves this nonlinearity then it is called maximally nonlinear.

Differentially k-Uniform: A mapping F: $\mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n}$ is called differentially k-uniform [95] if each equation

$$F(x+a) - F(x) = b, \text{ where } a \in \mathbb{F}_{2^n}, a \neq 0, b \in \mathbb{F}_{2^n},$$

has at most k solutions in $\mathbb{F}_{2^n}$.
For a differentially k-uniform S-box used in a block cipher, k must be low to ensure resistance against differential cryptanalysis. Differential cryptanalysis exploits the scenario where a particular input difference to an S-box leads to a particular output difference with high probability. It is clear from the definition above that if k is high such a difference will always exist. Note that k is always even and the minimum value of k is 2.

Almost Perfect Nonlinear (APN): F is called almost perfect nonlinear (APN) if it is differentially 2-uniform. [34]. It is known that if n is odd and F is maximally nonlinear then F is APN [17].

Lagrange Interpolation: F can be represented in the polynomial form as

$$F(x) = \sum_{i=0}^{2^n-1} b_i x^i, b_i \in \mathbb{F}_{2^n}.$$

Given the evaluation of $F(i)$ at all the 2^n inputs where $i = 0, 1, \cdots, 2^n - 1$, its polynomial representation can be computed by computing the polynomial coefficients b_i using the following lagrange interpolation formula [52]:

$$b_i = \begin{cases} F(0), & i = 0 \\ \sum_{\alpha \in \mathbb{F}_{2^n}} F(\alpha)\alpha^{-i}, & 1 \leq i \leq 2^n - 1 \end{cases}$$

The above formula only holds if $F(0) = 0$. If $F(0) \neq 0$ then we can represent the function in polynomial form as $F(x) = \sum_{i=0}^{2^n-1} d_i x^i$, where the coefficients d_i are gives as

$$d_i = \begin{cases} F(0), & i = 0 \\ \sum_{\alpha \in \mathbb{F}_{2^n}} F(\alpha)\alpha^{-i}, & 0 < i < 2^n - 1 \\ b_{2^n-1} + F(0), & i = 2^n - 1 \end{cases}$$

2.5 Sequences

Binary Sequence: $\underline{a} = \{a_i\}, a_i \in \mathbb{F}_2$, is a binary sequence over $\mathbb{F}_2$.

Periodic Sequences and Polynomial Functions: Let $\mathcal{S}$ be the set of all binary sequences with period $N \mid (2^n - 1)$ and $\mathcal{F}$ be the set of all polynomial functions from $\mathbb{F}_{2^n}$ to $\mathbb{F}_2$. Then for any sequence $\underline{a} = \{a_i\} \in \mathcal{S}$ there exists a polynomial function $f(x)$ in $\mathcal{F}$ such that

$$a_i = f(\alpha^i),\ i = 0, 1, 2, \cdots,$$

where α is the primitive element of $\mathbb{F}_{2^n}$. Note that $f(x)$ is called the trace representation of $\underline{a}$.

m-sequence: If $f(x)$ is a single trace term function with exponent t relatively prime to n, then its corresponding sequence is called an m-sequence, i.e.,

$$\underline{a} = Tr_1^n(\beta\alpha^i)\ i = 0, 1, 2, \cdots,\ \beta \in \mathbb{F}_{2^n}^*.$$

Note that m-sequence has period $2^n - 1$ and is generated by an n stage LFSR with a primitive feedback polynomial and a non-zero initial state.

Autocorrelation of Binary Sequences: Let $\underline{a}$ be a periodic binary sequence with period N, then the autocorrelation function of of $\underline{a}$ is defined as

$$C_{\underline{a}}(\tau) = \sum_{i=0}^{N-1} (-1)^{a_i + a_{i+\tau}}, 0 \leq \tau \leq N-1$$

If the autocorrelation function of the sequence $\underline{a}$ is

$$C_{\underline{a}}(\tau) = \begin{cases} N, & \text{if } \tau \equiv 0 \bmod N \\ -1, & \text{otherwise} \end{cases}$$

then $\underline{a}$ is said to have ideal two-level autocorrelation.
From a cryptographic view point two-level autocorrelation ensures that the attacker can not find correlations between shifted versions of the same keystream. The definition given above is for periodic autocorrelation however in a stream cipher the length of the keystream used is only a fraction of the total period. In that situation the attacker can only compute aperiodic autocorrelation. Analyzing aperiodic autocorrelation is a hard problem and it is believed that sequences with good periodic autocorrelation also have good aperiodic autocorrelation.

Orthogonal Functions: Let $f(x)$ be a polynomial function from $\mathbb{F}_{2^n}$ to $\mathbb{F}_2$ with $f(0)$=0. If we have

$$C_f(\lambda) = \sum_{x \in \mathbb{F}_{2^n}} (-1)^{f(\lambda x) + f(x)} = \begin{cases} 0, & \text{if } \lambda \neq 1 \\ 2^n, & \text{if } \lambda = 1 \end{cases}$$

where $\lambda \in \mathbb{F}_{2^n}$, then we say that $f(x)$ is orthogonal over $\mathbb{F}$. If $f(x)$ is the trace representation of a binary sequence $\underline{a}$ of period $2^n - 1$ and $C_{\underline{a}}(\tau)$ is the autocorrelation function of $\underline{a}$ then $C_f(\lambda)$ and $C_{\underline{a}}(\tau)$ are related as follows

$$C_f(\lambda) = 1 + C_{\underline{a}}(\tau)$$

where $\lambda = \alpha^\tau$. Also the trace representation of a binary sequence of period $2^n - 1$ with ideal two-level autocorrelation is an orthogonal function.

Linear Complexity of Binary Sequences: The linear complexity, LC, of a binary sequence is defined as the size in bits of the shortest LFSR required to generate that sequence.
The Berlekamp-Massey algorithm provides an efficient way of computing the linear complexity of a sequence [72]. For an m-sequence of length $2^n - 1$ this algorithm only requires $2n$ consecutive symbols to determine the feedback polynomial of the LFSR and hence the entire sequence. Therefore the keystream produced by a stream cipher must have large linear complexity.

Runs of Ones: For a binary string, λ consecutive ones (1's) preceded by zero and followed by zero is called a run of ones of length λ. We consider the runs of ones to be cyclic. For example 1100011110011111 has two (not three) cyclic runs of ones.

Chapter 3

WG Stream Ciphers

In this chapter we present WG family of stream ciphers which have designed randomness properties. The design of WG guarantees a keystream with ideal two-level autocorrelation, balance, long period, ideal tuple distribution, and high and exact linear complexity. We discuss how these properties are achieved by the proposed design and show how to select various parameters to obtain a stream cipher for the desired security level and efficient implementation. The security of WG against time/memory/data tradeoff attacks, algebraic attacks and correlation attacks, is evaluated. Finally we present WG-128 [87] as a concrete example of a WG stream cipher with a key size of 128 bits.

3.1 Background

Traditionally many hardware oriented stream ciphers have been built using linear feedback shift registers (LFSRs) and Boolean functions with compact Algebraic Normal Forms (ANFs). This allowed for a very small and efficient implementation in hardware. However following the discovery of algebraic attacks [20, 21, 75, 5], using Boolean functions with compact ANFs is no longer secure. One way to defeat the algebraic attacks is to use nonlinear feedback shift registers (NFSRs) or more generally, update the state of the stream cipher nonlinearly. Many hardware oriented stream ciphers in the eSTREAM project including F-FCSR, Grain, Mickey,

and Trivium, have been designed according to this approach [36]. Whereas the tools to analyze the LFSR based stream cipher designs are well developed, the same is not true for NFSRs. Some results exist for a class of NFSR's known as feedback with carry shift registers (FCSR) [61, 62, 63] but their use in the design of stream ciphers is relatively new. Therefore the security of most of these stream ciphers rely on the difficulty of analyzing the design itself. An alternative approach is to design a stream cipher in such a way that it is very easy to analyze. This allows the designers to explicitly prove various security properties of the design. The WG family of stream ciphers have been designed using this approach. To defeat the algebraic attacks on LFSR based stream ciphers, WG relies on nonlinear Boolean functions with large number of inputs, high degree and complex ANF forms. To overcome the implementation complexity of the Boolean function it has been designed in polynomial form instead of ANF form. Implementation of such functions require small finite field multipliers. This results in a moderate increase in the hardware requirements, however the design is still practical.

A WG stream cipher consists of a WG keystream generator which produces a long pseudo-random keystream. The keystream is XORed with the plaintext to produce the ciphertext. From here onwards we will focus on the design and randomness properties of the WG keystream generators. The WG keystream generators use Welch-Gong(WG) transformations as the filtering functions. The WG transformations have very large ANFs and can be implemented in polynomial form using finite field arithmetic. These transformations correspond to the WG transformation sequences that were discovered by Golomb, Gong, and Gaal in 1998 [94]. In 2002, Gong and Youssef presented several cryptographic properties of WG transformation sequences. These properties make them an ideal candidate for cryptographic use.

This chapter is organized as follows. In Section 3.2 we first show that it is impractical to use WG transformation sequence generators from [54, 94] as keystream generators in practical stream ciphers. We then examine the possibility of using a small WG transformation as a filtering function in a nonlinear filter to build a keystream generator. In Section 3.3 we show which cryptographic properties of the

WG transformation sequences can be preserved in this design. For the properties that can not be preserved completely we list the possible tradeoffs. Then we show how various parameters of the designs can be selected to design WG stream ciphers for various security levels and application requirements. Finally in Section 3.5 we present WG-128, as a concrete example of a WG keystream generator with a secret key of size 128 bits. Note that WG-128 is a slightly modified version of the WG stream cipher which is a phase 2 candidate in profile 2 of the ECRYPT stream cipher project: eSTREAM [36].

3.2 WG Transformation and Stream Ciphers

3.2.1 WG Transformation

In this section we give a formal definition of WG transformation taken from [94]. Let $n \neq 0 \bmod 3$, and $t(x) = x + x^{q_1} + x^{q_2} + x^{q_3} + x^{q_4}$, $x \in \mathbb{F}_{2^n}$, where q_i's are defined as:
For $n = 3k - 1$

$$\begin{aligned} q_1 &= 2^k + 1, \\ q_2 &= 2^{2k-1} + 2^{k-1} + 1, \\ q_3 &= 2^{2k-1} - 2^{k-1} + 1, \\ q_4 &= 2^{2k-1} + 2^k - 1, \end{aligned} \tag{3.1}$$

and for $n = 3k - 2$

$$\begin{aligned} q_1 &= 2^{k-1} + 1, \\ q_2 &= 2^{2k-2} + 2^{k-1} + 1, \\ q_3 &= 2^{2k-2} - 2^{k-1} + 1, \\ q_4 &= 2^{2k-1} - 2^{k-1} + 1. \end{aligned} \tag{3.2}$$

Then a function

$$f(x) = \mathrm{Tr}(t(x+1)+1), x \in \mathbb{F}_{2^n} \tag{3.3}$$

is called the WG transformation. Note that $f(x)$ is a function from $\mathbb{F}_{2^n} \rightarrow \mathbb{F}_2$. Let α be a primitive element of $\mathbb{F}_{2^n}$. Let $\underline{w} = w_i$ where

$$w_i = f(\alpha^i) = Tr(t(\alpha^i + 1) + 1), i = 0, 1, \ldots$$

Then $\underline{w}$ is called a WG sequence.

In [54] Gong et al. discussed various cryptographic properties of the binary WG sequence $\underline{w}$. The WG sequence has period $2^n - 1$, is balanced and has ideal two-level autocorrelation. The linear complexity of the WG sequence ($n(2^{\lceil n/3 \rceil} - 3)$) increases exponentially with n and its cross correlation with an m-sequence is only three valued. Furthermore if the basis used for computation of the sequence in $\mathbb{F}_{2^n}$ is chosen properly, the Boolean function that corresponds to the the WG transformation is also 1-resilient, i.e., the output of the WG transformation is not correlated to any single input. Note that it is very difficult to design sequences which have all the above cryptographic properties. Therefore the WG sequences are well-suited for use as pseudo-random sequences in cryptography.

3.2.2 Practical Stream Cipher Design

Despite its cryptographic properties, generating WG sequence $\underline{w}$ is not very efficient when it comes to practical cryptographic applications such as stream ciphers. Most cryptographic applications require a secret key to be at least 128 bits. Also there is a consensus among the designers of stream ciphers that the internal state of the cipher must at least be twice the size of the secret key. This would require a WG sequence generator to have an internal state of 256 bits. From Eqn. (3.3) it is evident that a straight forward implementation of the WG sequence generator would require computations in $\mathbb{F}_{2^{256}}$ making it highly inefficient. In [54], the authors also mentioned the possibility of using the WG transformation as a combining function in combinatorial function generator or as a filtering function in filtering generator. However this approach was not explored further. There are several questions that

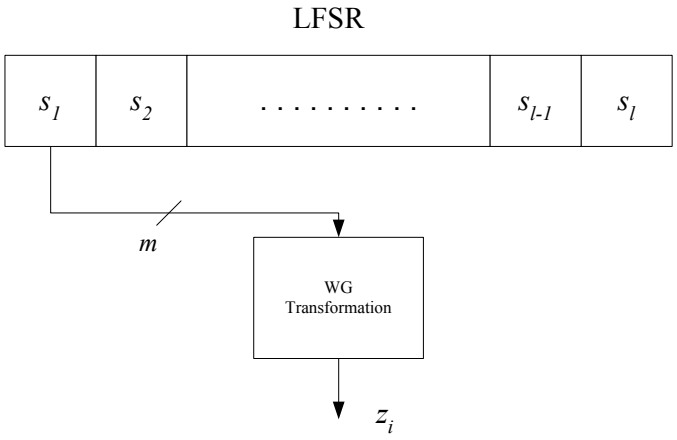

Figure 3.1: Block diagram of WG keystream generator

must be answered to evaluate the security and performance of such generators:

- Is it possible to design the generator such that its output sequence has cryptographic properties similar to those of WG sequence $\underline{w}$?
- Which properties of the WG sequence can be preserved completely and for the properties that can not be preserved completely what are the possible tradeoffs?
- Can such generators provide the security and efficiency expected of a stream cipher?

In this chapter we answer the above questions for a filter generator which uses WG transformation as the filtering function. We show that by using a non-binary LFSR and a small WG transformation, we can design a practical keystream generator that can be used as a stream cipher. We also show how to choose the design parameters to ensure that the generator has cryptographic properties of the WG sequence.

3.3 WG Keystream Generator Family

We first give an overview of the WG keystream generator family. A WG keystream generator can be regarded as a nonlinear filter (see Figure 3.1). It consists of an

l stage LFSR with a primitive feedback polynomial over $\mathbb{F}_{2^m}$. The LFSR generates a maximal length sequence over $\mathbb{F}_{2^m}$ which is filtered by a, $\mathbb{F}_{2^m} \rightarrow \mathbb{F}_2$, WG transformation to generate the binary keystream. This simple design generates a keystream which has period $2^n - 1$, where $n = lm$ and is easy to analyze for various cryptographic properties. If the feedback polynomial of the LFSR and the parameters of the WG transformation are selected appropriately, the generator will be efficient and the keystream will inherit most cryptographic properties of the WG transformation sequence $\underline{w}$.

3.3.1 WG Parameter Selection

In this section we show how to select various parameters of the WG keystream generator. First we give a list of all the parameters and their relation to the desired cryptographic properties. Then we outline the algorithms to find these parameters.

- For the $\mathbb{F}_{2^m} \rightarrow \mathbb{F}_2$ WG transformation, $m \neq 0 \bmod 3$ (see the definition of WG transformation in Section 3.2).

- The efficiency of the WG keystream generator and most of its cryptographic properties depend on the basis chosen for the computation of the WG transform. Computation of $t(x)$ (Eqn. (3.3)) in the WG transformation involves exponentiations in $\mathbb{F}_{2^m}$. If the field elements are represented in normal basis, then raising an element to a power of 2 is as simple as a cyclic shift. Since the exponents in $t(x)$ involves such computations using normal basis is more efficient than polynomial basis.

- The complexity of computing the WG transformation mainly resides in the multiplications in $\mathbb{F}_{2^m}$. The hardware implementation of a normal basis multiplier in $\mathbb{F}_{2^m}$ depends on the basis used to represent the field elements. The normal basis for which the hardware complexity of the normal basis multiplier is minimum is known as the optimal normal basis (ONB). Using ONB can further improve the hardware efficiency of the generator, however, this requirement may not always be fulfilled as ONB do not exist for every m.

Also note that the generator can always be implemented using a polynomial basis although it may increase the implementation complexity.

- It must be ensured that for the selected basis, the Boolean function that corresponds to the WG transformation is at least 1-order resilient. Otherwise a different basis must be selected to ensure the resiliency of WG transformation.

- The most distinguishing feature of the WG keystream generator is the periodic two-level autocorrelation of its keystream. It is known that this property of the WG transform sequence $\underline{w}$ is inherited by the WG keystream generator. Here we reproduce the proof for completeness. Let $\{a_i\}$ be the output of the generator:

$$\begin{aligned} a_i &= f(b_i), i = 0, 1, \ldots \text{ and} \\ u(x) &= f \circ Tr_m^n(x), \end{aligned} \tag{3.4}$$

  where $u(x)$ is the composition of $Tr_m^n(x)$ and f. Note that $\{b_i\}$ is the m-sequence generated by the LFSR over $\mathbb{F}_{2^m}$ with the trace representation $Tr_n^m(x)$ and f is the WG transformation. It is known [52] that if f is an orthogonal function then the sequence that corresponds to u has two-level autocorrelation. Since the WG transformation f is an orthogonal function [94, 32] the keystream generated by the generator has periodic two-level autocorrelation. To ensure that the keystream has this property the finite field computations (in $\mathbb{F}_{2^m}$) in the feedback polynomial of the LFSR and the WG transformation must use the same basis.

- The linear complexity of the keystream depends on two parameters: n and m. Therefore n and m must be selected to ensure that linear complexity is sufficiently high. The exact formula for the linear complexity and tradeoff between n and m are discussed in Section 3.3.5.

- Some general restrictions that apply to all keystream generators must also be met. For example the size of the internal state should be at least twice the

size of the secret key. The algebraic degree of the filtering function and its algebraic immunity must be high to prevent algebraic attacks and so on.

Now we show how to find various parameters and design a secure and efficient WG keystream generator.

3.3.2 Finding Optimal Normal Basis in $\mathbb{F}_{2^m}$

The number of normal bases in an extension field is usually quite large and therefore one basis can be found easily. Here we provide an outline of an algorithm to find an Optimal Normal Basis (ONB) in $\mathbb{F}_{2^m}$. First we state the following fact:

Fact 1 [79]. *Let δ be a primitive $(2m+1)th$ root of unity in $\mathbb{F}_{2^{2m}}$ and $\gamma = \delta + \delta^{-1}$. If 2 is primitive in $\mathbb{Z}_{2m+1}$ or $2m+1$ is a prime congruent to 3 modulo 4 and 2 generates the quadratic residues in $\mathbb{Z}_{2m+1}$ then γ is an element of the subfield $\mathbb{F}_{2^m}$ and $N = [\gamma, \gamma^{2^1}, \gamma^{2^2}, \cdots, \gamma^{2^{m-1}}]$ is an ONB of the subfield $\mathbb{F}_{2^m}$.* We select the ONB through the following steps:

- Construct the finite field, $\mathbb{F}_{2^{2m}}$, defined by a primitive polynomial of degree $2m$ over $\mathbb{F}_2$. Let α be the root of this polynomial. Find a primitive $(2m+1)st$ root of unity, δ, in this field and compute δ^{-1}. Now $\gamma = \delta + \delta^{-1}$.

- We have γ in $\mathbb{F}_{2^{2m}}$. According to Fact 1, γ also belongs to $\mathbb{F}_{2^m}$. In order to map γ to this subfield we define $d = \frac{2^{2m}-1}{2^m-1}$ and $\eta = \alpha^d$. Then η is a primitive element of $\mathbb{F}_{2^m}$.

- Find the minimum polynomial of η, say $g(x)$, given by

$$g(x) = \prod_{i=0}^{m-1} (x - \eta^{2^i}). \tag{3.5}$$

  $g(x)$ is a polynomial of degree m over $\mathbb{F}_2$. Use $g(x)$ as the defining polynomial of $\mathbb{F}_{2^m}$. Let β be the primitive root of $g(x)$. Now we only need to define γ in terms of β.

- To find γ in terms of β we need the following computation in $\mathbb{F}_{2^{2^m}}$. Find i, $\{i \mid \gamma = \eta^i, 1 \leq i \leq 2^m - 2\}$. Once we know the desired i we can write $\gamma = \beta^i$ in $\mathbb{F}_{2^m}$.
- The ONB is now simply given as $\{\gamma, \gamma^{2^1}, \gamma^{2^2}, \cdots, \gamma^{2^{m-1}}\}$.

3.3.3 Resiliency of WG Transformation

The Boolean function that corresponds to the WG transformation depends on the basis used for the computation of WG transformation. To find the basis that give a resilient Boolean function we refer to [54]. First define the following terms:
Let $c = e^{-1} \bmod 2^m - 1$, where $e = 2^{2t} - 2^t + 1$, for $3t = 1 \bmod m$. Let

$$D = \{x \in \mathbb{F}_{2^m}^* \mid Tr_1^n(x^c) = 0\}. \tag{3.6}$$

where $\mathbb{F}_{2^m}^*$ is the multiplicative group of $\mathbb{F}_{2^m}$. Also let

$$R_{\underline{\gamma}} = \{(Tr_1^m(\lambda\gamma_0), \cdots, (Tr_1^m(\lambda\gamma_{m-1})) \in \mathbb{F}_{2^m} \mid \lambda \in D\} \tag{3.7}$$

where $(\gamma_0, \gamma_1, \cdots, \gamma_{m-1})$ is a basis of $\mathbb{F}_{2^m}$. Note that this basis is the same as selected in Section 3.3.2. ONB is chosen for an efficient implementation however the generator can be designed using a different basis as well. Now the necessary condition for the resiliency of WG transformation is given in the following fact.

Fact 2 [54]. Let $f(\underline{x})$ be the Boolean form of the WG transformation $f(x)$. Then $f(\underline{x})$ is r-resilient if and only if all vectors in $W_r = \{\underline{w} \mid 1 \leq H(\underline{w}) \leq r\}$ appear in $R_{\underline{\alpha}}$, i.e., $W_r \subset R_{\underline{\gamma}}$.

Since our design is a simple nonlinear filter, 1-order resiliency is sufficient for the WG transformation. To check whether a selected basis $\{\gamma_0, \gamma_1, \cdots, \gamma_{m-1}\}$ gives a Boolean function with 1-order resiliency, start by computing the set D. Note that $|D| = 2^{m-1} - 1$. For each member of D compute the m-bit vector from Eqn. (3.7) which gives the set $R_{\underline{\gamma}}$. If set $R_{\underline{\gamma}}$ contains all m-bit distinct vectors with Hamming weight exactly 1, the Boolean form of the WG transformation is 1-order resilient.

Otherwise discard the selected basis, select a new basis and check the resiliency of the WG transformation under the new basis. Repeat the procedure until a proper basis is found. It must be noted that since m is small and the number of bases for which WG transformation is resilient is very large, finding a proper basis is not difficult.

3.3.4 Selection of LFSR Feedback Polynomial

The LFSR feedback polynomial should be a primitive polynomial of degree l over $\mathbb{F}_{2^m}$. For the keystream to have two-level autocorrelation property, the computations in the feedback polynomial must be in the same basis as the WG transformation. To ensure the above we select the feedback polynomial in the following way:

- Construct $\mathbb{F}_{2^m}$ from the defining polynomial $g(x)$, where β is the root of $g(x)$.
- γ defines the basis selected above and is therefore known. Pick a polynomial of degree l with constant term as γ and all the other coefficients as either 0 or 1. Check if the selected polynomial is primitive over $\mathbb{F}_{2^m}$. If not try a different polynomial until a primitive polynomial is found.

3.3.5 Achieving Desired Linear Complexity

As shown in Eqn. (3.4), the keystream of the generator can be regarded as composition of the WG transformation and an m-sequence. Sequences generated in such manner are also known as generalized GMW sequences [52]. The linear complexity of the generalized GMW sequences can be determined exactly (see [52, page 66]). Let us consider the $\mathbb{F}_{2^m} \rightarrow \mathbb{F}_2$ WG transformation, f. From [94],

$$f(x) = \sum_{i \in I} Tr_1^n(x^i) \tag{3.8}$$

where $I = I_1 \cup I_2$ and for $m = 3k - 1$,

$$I_1 = \{2^{2k-1} + 2^{k-1} + 2 + i \mid 0 \leq i \leq 2^{k-1} - 3\},$$
$$I_2 = \{2^{2k} + 3 + 2i \mid 0 \leq i \leq 2^{k-1} - 2\},$$

and for $m = 3k - 2$

$$I_1 = \{2^{k-1} + 2 + i \mid 0 \leq i \leq 2^{k-1} - 3\},$$
$$I_2 = \{2^{2k-1} + 2^{k-1} + 2 + i \mid 0 \leq i \leq 2^{k-1} - 3\}.$$

Let LC_{WGK} be the linear complexity of the WG keystream generator then from Eqn. (3.8) and Eqn. (3.15) [52, page 66]

$$LC_{WGK} = m \times \sum_{i \in I} l^{w(i)}. \tag{3.9}$$

The linear complexity of the WG keystream generator is less than the linear complexity of the WG transformation sequence with period $2^n - 1$. Some linear complexity is sacrificed by using a smaller WG transformation(i.e., $\mathbb{F}_{2^m} \rightarrow \mathbb{F}_2$ instead of $\mathbb{F}_{2^n} \rightarrow \mathbb{F}_2$) to achieve an efficient implementation. Note that linear complexity of the WG keystream generator depends on n and m. If m, which determines the size of WG transformation is fixed, linear complexity can be increased by increasing the number of stages in the LFSR and vice versa.

3.3.6 Internal State and Degree of WG Transformation

To provide security against common attacks on stream ciphers, the keystream generator must meet some additional conditions. If K is the size of the secret key in bits, then to provide protection against time/memory/data tradeoff attacks, the size of the internal state (or length of LFSR in bits) must be at least $2K$, i.e., $n \geq 2K$. To provide protection against algebraic attacks the algebraic degree of the WG transformation should be sufficiently high and it must not have any low degree approximations [23]. The algebraic degree of the WG transformation in m variables is given by $\lceil m/3 \rceil + 1$. Then according to [23] the complexity of the alge-

braic attack that recovers the secret key K of the WG generator is approximately $7/64 \cdot \binom{n}{\lceil m/3 \rceil + 1}^{log_2 7}$ word operations. Therefore n and m must be chosen to keep this complexity sufficiently high. However note that if a low degree approximation of the WG transformation is found, the complexity of the algebraic attack will be reduced. A meaningful low degree approximation of the WG transformation means an approximation with degree less than the degree of WG transformation itself. We have experimantally verified that WG transformations in 11, 13 and 14 variables (which have algebraic degrees 5, 6 and 6 respectively) do not have low any degree approximations. Note that the degree of the WG transformation is significantly less than m. Therefore from the assertion made in [75] and our experimental results, on WG transformation in 11, 13 and 14 variables, we conjecture that the probability of the existence of such low degree approximation is negligible. Therefore we can say that to prevent algebraic attacks, n and m must be selected so that

$$7/64 \cdot \binom{n}{\lceil m/3 \rceil + 1}^{log_2 7} > 2^K.$$

3.4 Application Specific Design of WG

To design a WG keystream generator parameters m and l can be chosen according to the target application. Table 3.1 shows various possible WG keystream generators. Note that n is the size of the internal state (or LFSR) in bits, LC_{WGK} is the linear complexity of the generator and C_{AA} is the complexity of the algebraic attack on the generator. A 'Y' in the column ONB indicates that optimal normal basis exist for the corresponding m and a 'N' indicates that it does not exist. All the designs in Table 3.1 allow a secret key size of at least 80 bits. The linear complexity of the generator can be selected according to the application requirements. For example if the generator is to be used for encryption and decryption of packet based data a very large linear complexity of the keystream may not be required. Consider the design with $m = 16$, $l = 16$ and secret key K of length 100 bits. The linear complexity in this case is $2^{32.5}$ and therefore the maximum amount of keystream that can be generated with one pair of secret key K and an initial vector

Table 3.1: WG keystream generators and their corresponding security levels

m	l	n	LC_{WGK}	C_{AA}	ONB
11	16	176	$2^{24.7}$	$2^{81.7}$	Y
11	24	264	$2^{27.6}$	2^{90}	Y
11	32	352	$2^{29.6}$	$2^{95.8}$	Y
14	16	224	$2^{29.2}$	$2^{101.1}$	Y
14	24	336	$2^{32.6}$	2^{111}	Y
14	32	448	2^{35}	$2^{118.1}$	Y
16	10	160	$2^{28.1}$	$2^{105.3}$	N
16	16	256	$2^{32.5}$	$2^{118.8}$	N
16	24	384	$2^{36.4}$	$2^{130.4}$	N
11	32	512	$2^{39.3}$	$2^{138.6}$	N
29	11	319	2^{45}	2^{182}	Y

is approximately 2^{32} (or 4 gigabits). Since the packet size in most packet based communications applications is much smaller (a few bytes to a few kilobytes) and the generator is re-initialized with a new initial vector for each packet, this linear complexity is sufficiently high. Although ONB do not exist for $m = 16$, the WG transformation can be implemented efficiently by using the subfield decomposition of $\mathbb{F}_{2^{16}}$ [96]. A specific implementation of inversion in $\mathbb{F}_{2^{16}}$ is also given in [14]. Also note that for packet based applications the re-initialization time of the generator is very critical. We will show in the next section that WG keystream generators can be re-initialized in only $2l$ steps. If an application requires a long keystream, then a larger linear complexity can be obtained by increasing the size of l or m or both. For example for $m = 29$ and l=11 the generator can easily handle a keystream of length 2^{45} bits.

3.5 WG-128 Keystream Generator

Now we give a concrete example of a 128-bit WG keystream generator: WG-128. We derive all the parameters of the generator using the procedures given in the previous section. We also show how to implement this generator and discuss its

security. The WG-128 keystream generator is shown in Figure 3.1. The generator allows for a 128-bit secret key and 128-bit initial vectors(IVs). It consists of an 11 stage linear feedback shift register(LFSR) over $\mathbb{F}_{2^{29}}$. The feedback polynomial of the LFSR is primitive over $\mathbb{F}_{2^{29}}$ and is given by

$$p(x) = x^{11} + x^{10} + x^9 + x^6 + x^3 + x + \gamma \tag{3.10}$$

where $\gamma = \beta^{464730077}$ and β is a root of $g(x)$:

$$\begin{aligned} g(x) &= x^{29} + x^{28} + x^{24} + x^{21} + x^{20} + x^{19} + x^{18} + x^{17} + \\ &\quad x^{14} + x^{12} + x^{11} + x^{10} + x^7 + x^6 + x^4 + x + 1. \end{aligned} \tag{3.11}$$

We define $S(1), S(2), S(3), .., S(11) \in \mathbb{F}_{2^{29}}$ to be the state of the LFSR and the internal state of the WG keystream generator. We denote the output of the LFSR as $b_i = S(11 - i), i = 0, 1, .., 10$. Then for $i \geq 11$, we have

$$b_i = b_{i-1} + b_{i-2} + b_{i-5} + b_{i-8} + b_{i-10} + \gamma b_{i-11}, i \geq 11 \tag{3.12}$$

The output of the LFSR is filtered by a nonlinear WG transformation, $\mathbb{F}_{2^{29}} \rightarrow \mathbb{F}_2$, to produce the keystream. All the elements of $\mathbb{F}_{2^{29}}$ are represented in normal basis and all the finite field computations are performed in normal basis as well.

For WG transformation, consider $\mathbb{F}_{2^{29}}$ generated by the primitive polynomial $g(x)$ with β as its root. Let

$$t(x) = x + x^{2^{10}+1} + x^{2^{19}+2^9+1} + x^{2^{19}-2^9+1} + x^{2^{19}+2^{10}-1}, x \in \mathbb{F}_{2^{29}}. \tag{3.13}$$

We can rewrite $t(x)$ as

$$t(x) = x + x^{2^{10}+1} + x^{2^{19}+2^9+1} + x(x^{2^{10}-1})^{2^9} + x^{2^{19}}(x^{2^{10}-1}), x \in \mathbb{F}_{2^{29}}. \tag{3.14}$$

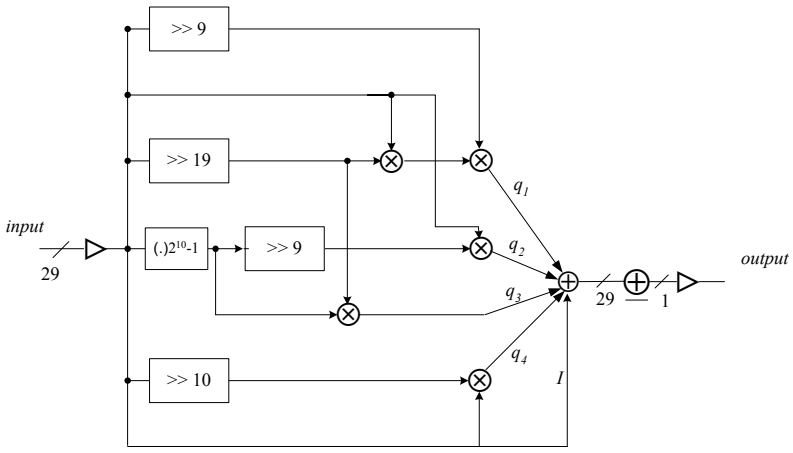

Figure 3.2: Implementation of WG transformation: $\mathbb{F}_{2^{29}} \rightarrow \mathrm{F}_2$

Then the WG transformation is given as

$$f(x) = \mathrm{Tr}(t(x+1)+1), x \in \mathbb{F}_{2^{29}}. \tag{3.15}$$

Note that $Tr(1) = 1$, therefore we can rewrite $f(x)$ as

$$f(x) = \mathrm{Tr}(t(x+1)) + 1, x \in \mathbb{F}_{2^{29}}. \tag{3.16}$$

3.5.1 Implementation of WG-128

We first state a few facts about computation in $\mathbb{F}_{2^{29}}$ when field elements are represented in normal basis.

- If the field elements are represented in a normal basis, squaring can be done by right cyclic shift.
- If γ is the generator of the normal basis then $1 = \sum_{i=0}^{28} \gamma^{2^i}$, i.e., the 29 bit vector representing 1 is all ones vector. Therefore addition of a field element, in normal basis representation, with 1 can be done by simply inverting the bits of that element.

- The trace of all the basis elements is one, i.e., $\mathrm{Tr}(\gamma^{2^i}) = 1$ where $0 \leq i \leq 28$. Therefore the trace of any field element represented as a 29 bit vector can be obtained by adding all the bits of the elements over $\mathbb{F}_2$ (i.e., XOR).

To facilitate the implementation of the WG transformation we provide a more specific description in Figure 3.2. Note that Figure 3.2 comes directly from Eqns. (3.14), (3.16), and the facts stated above. From the figure the output of the WG transformation can be written as

$$output = \triangleright\underline{\bigoplus}((q_1 \oplus (q_2 \oplus (q_3 \oplus (q_4 \oplus I))))) \tag{3.17}$$

where

$$q_1 = (I \gg 9) \otimes ((I \gg 19) \otimes I)$$

$$q_2 = (I^{2^{10}-1} \gg 9) \otimes I$$

$$q_3 = I^{2^{10}-1} \otimes (I \gg 19)$$

$$q_4 = (I \gg 10) \otimes I$$

$$I = \triangleright(input).$$

The notation $x \otimes y$ means normal basis multiplication of x and y in $\mathbb{F}_{2^{29}}$ defined by $g(x)$. Similarly $(x)^{2^{10}-1}$ means raising x to the power $2^{10} - 1$ in $\mathbb{F}_{2^{29}}$ defined by $g(x)$. Note that this calculation requires 4 multiplications in $\mathbb{F}_{2^{29}}$. $x \oplus y$ represents the bitwise addition (XOR) of words x and y, and $x \gg c$ represents the cyclic shift of x, c stages to the right where c is a positive integer. The symbol $\triangleright(x)$ means all the 29 bits of x are complemented and $\underline{\bigoplus}(x)$ means the addition of the 29 bits of x over $\mathbb{F}_2$ (XOR) i.e., for $x = (x_0, .., x_{28})$, $\underline{\bigoplus}(x) = \sum_{i=0}^{28} x_i \bmod 2$.

3.5.2 Key Initialization and Re-synchronization

The state of the LFSR is represented as $S(1), S(2), S(3), .., S(11) \in \mathrm{F}_{2^{29}}$; Each stage $S(i) \in \mathrm{F}_{2^{29}}$, is represented as $S_{1,..,29}(i)$ where $1 \leq i \leq 11$. Similarly we represent the key bits as $k_{1,..,j}, 1 \leq j \leq 128$ and IV bits as $IV_{1,..,m}, 1 \leq m \leq 128$. The key and the IV are loaded into the LFSR as follows:

$S_{1,..16}(1) = k_{1,..,16}$ $S_{17,..24}(1) = IV_{1,..,8}$ $S_{1,..8}(2) = k_{17,..,24}$
$S_{9,..24}(2) = IV_{9,..,24}$ $S_{1,..16}(3) = k_{25,..,40}$ $S_{17,..24}(3) = IV_{25,..,32}$
$S_{1,..8}(4) = k_{41,..,48}$ $S_{9,..24}(4) = IV_{33,..,48}$ $S_{1,..16}(5) = k_{49,..,64}$
$S_{17,..24}(5) = IV_{49,..,56}$ $S_{1,..8}(6) = k_{65,..,72}$ $S_{9,..24}(6) = IV_{57,..,72}$
$S_{1,..16}(7) = k_{73,..,88}$ $S_{17,..24}(7) = IV_{73,..,80}$ $S_{1,..8}(8) = k_{89,..,96}$
$S_{9,..24}(8) = IV_{81,..,96}$ $S_{1,..16}(9) = k_{97,..,112}$ $S_{17,..24}(9) = IV_{97,..,104}$
$S_{1,..8}(10) = k_{113,..,120}$ $S_{9,..24}(10) = IV_{105,..,120}$ $S_{1,..8}(11) = k_{121,..,128}$
$S_{17,..24}(11) = IV_{121,..,128}$

All the remaining bits of the LFSR are set to zero. Once LFSR has been loaded with the key and IV, the keystream generator is run for 22 clock cycles. This is the key initialization phase of the cipher operation. During this phase the 29 bit vector, given by

$$keyinitvec = ((q_1 \oplus (q_2 \oplus (q_3 \oplus (q_4 \oplus I)))))) \tag{3.18}$$

in Figure 3.2, is added to the feedback of the LFSR which is then used to update the LFSR. The key initialization process is shown in Figure 3.3. Once the key has been initialized the LFSR is clocked once and the 1 bit output of the WG transformation gives the first bit of the running keystream. The linear complexity of the keystream is slightly more than 2^{45} (from Eqn. (3.9)), and therefore the maximum length of the keystream allowed to be generated with a single key and IV is 2^{45}. After this the cipher must be reinitialized with a new IV or a new key or both.

3.5.3 Security of WG-128

The keystream generated by the WG keystream generator is balanced, has period $2^{319} - 1$, and has ideal two-level autocorrelation. The linear complexity of the keystream is approximately $2^{45.0415}$ and it has ideal t-tuple distribution where $1 \leq t \leq 11$. The Boolean function that corresponds to $\mathbb{F}_{2^{29}} \rightarrow \mathbb{F}_2$ WG transformation has algebraic degree 11 and its nonlinearity is $2^{28} - 2^{14} = 268419072$. The size of the WG internal state is 319 bits which is more than twice the largest possible key size, therefore it is secure against time/memory/data tradeoff attacks. The complexity

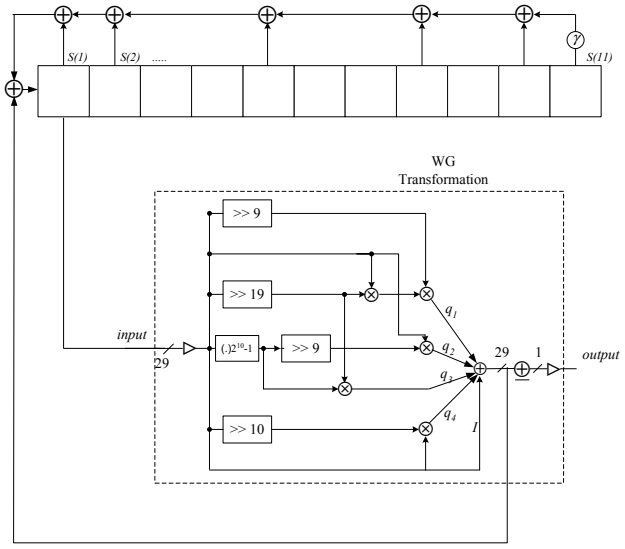

Figure 3.3: Key initialization phase of WG cipher

of the algebraic attacks as explained in Section 3.3.6 is approximately 2^{182} which is much higher than the largest possible key space.

To analyze the resistance of WG stream cipher against correlation attacks we have to consider the nonlinearity and resiliency of the WG transformation. The WG transformation in WG-128 is 1-order resilient, i.e., the output of the WG transformation or the keystream is not correlated to any single input bit of the LFSR output.This suggests that WG cipher is secure against correlation attacks. However we must consider more sophisticated correlation attacks in which WG transformation is approximated by linear functions. These linear approximations can then be used to derive a generator matrix of a linear code and its decoding can be performed by a Maximum Likelihood (ML) decoding algorithm to recover the internal state of LFSR. The exact details of the attack are beyond the scope of this thesis and the reader is referred to [19] for more details. We will only use theoretical bounds derived in [19] to make sure that our cipher is secure against these attacks. Let f' be the Boolean function representation of WG transformation and l be a linear function with minimum Hamming distance to f'. Then the probability that

they produce the same output for a given input x is given by

$$P(f'(x) = l(x)) = \frac{2^{29} - (2^{28} - 2^{14})}{2^{29}} = 0.5000305. \tag{3.19}$$

Using the results given in [19] the amount of keystream required for a successful attack is given by

$$N \approx (k \cdot 12 \cdot ln2)^{1/3} \cdot \epsilon^{-2} \cdot 2^{\frac{319-k}{3}} \tag{3.20}$$

and decoding complexity is given by

$$C_{dec} = 2^k \cdot k \cdot \frac{2ln2}{(2\epsilon)^6}, \tag{3.21}$$

where $\epsilon = P(f'(x) = l(x)) - 0.5 = 0.0000305$ and k is the number of LFSR internal state bits recovered. If we choose k to be very small, i.e., $k = 5$, the amount of keystream required for the attack is approximately 2^{133}, which is not a realistic amount to collect. Moreover the complexity of pre-computation phase is more than 2^{266}. Since the maximum keystream that can be produced with a single key and IV is 2^{45} we choose $k = 274$ to reduce N to this number. Now the complexity of the decoding phase is approximately 2^{366}. This analysis shows that the WG cipher is secure against this kind of correlation attacks.

3.5.4 Hardware Implementation of WG-128

The normal basis multiplication in $\mathbb{F}_{2^{29}}$ is the most expensive operation in the hardware implementation of WG-128. Depending on the type and number of multipliers used, a wide variety of area versus speed tradeoffs are possible. An implementation can range from using a single serial normal basis multiplier to multiple parallel normal basis multipliers. While the underlying platform and speed requirements will dictate an implementation, we suggest a straightforward implementation which can achieve high speed with relatively less amount of hardware. A parallel normal basis multiplier multiplies two elements in normal basis and produces the result in one clock cycle. Therefore we suggest using 2 parallel normal basis multipliers. In

Figure 3.2, there are nine multiplications in the WG transformation and one in the feedback of the LFSR. Therefore a keystream bit can be produced every 5th clock cycle. Note that the cyclic shifts can be done by rearranging the connections to the registers or the inputs of multipliers.

Several normal basis multipliers have been proposed in the literature [71, 113, 111, 102, 1, 103, 104]. As stated earlier the hardware complexity of a normal basis multiplier depends on the basis used to represent the field elements. More precisely it depends on the multiplication matrix C_N of the chosen normal basis [79]. The complexity is directly proportional to the number of ones in the C_N matrix. To facilitate the hardware implementation of a normal basis multiplier for WG transformation we provide the C_N matrix that corresponds to the chosen optimal normal basis of $\mathbb{F}_{2^{29}}$ in Appendix A. The optimal normal basis multiplier given in [111] can be implemented with 841 AND gates and 1218 XOR gates only. Note that this is the best multiplier known to us presently. Any improvement in the efficiency of normal basis finite field multiplier can increase the efficiency of the hardware implementation of the WG keystream generator further.

3.5.5 Software Implementation

The software implementation of the WG cipher is straightforward. Since the elements of $\mathbb{F}_{2^{29}}$ can be represented within a single word of a 32 bit processor, each of the addition, negation and shift operations can be performed by a single instruction. The time consuming operations in software are the normal basis multiplications. The exact implementation of the multiplier depends on the multiplication matrix. Several algorithms exist for performing normal basis multiplication in software efficiently [92, 105]. We provide C implementations of the normal basis multiplication and inversion [41] algorithms for the optimal normal basis over $\mathbb{F}_{2^{29}}$ in Appendix A. The implementations are provided as examples and we do not claim them to be the most efficient. Any normal basis multiplication and inversion algorithm can be used for the software implementation of the cipher.

3.6 Conclusions

In this chapter we presented a family of keystream generators with ideal two-level autocorrelation and other guaranteed randomness properties. We showed how WG keystream generators can be designed to fulfil various application requirements. Finally we presented WG-128 as a concrete example from the WG family and evaluated its security and efficiency.

Chapter 4

RC4-like Keystream Generator

In Chapter 3 we proposed a stream cipher suitable for hardware applications. In this chapter we propose a new and efficient stream cipher suitable for software applications. The cipher is similar in design to the famous and widely used RC4 stream cipher and is therefore named RC4-like keystream generator. The proposed generator produces 32 or 64 bits in each iteration and can be implemented with reasonable memory requirements. It has a huge internal state and offers higher resistance to state recovery attacks than the original 8-bit RC4. Further, on a 32-bit processor the generator is 3.1 times faster than original RC4. We also show that it can resist attacks that are successful on the original RC4. The generator is suitable for high speed software encryption.

4.1 Background

RC4 was designed by Ron Rivest in 1987 and kept as a trade secret until it leaked out in 1994. Unlike many stream ciphers that are based on LFSRs, the design of RC4 is based on exchange-shuffle paradigm [66]. The design consists of a large table/array that slowly changes with time under its own control. The popularity of the design is due to its simplicity and ease of implementation. For the generators based on such designs it is hard to theoretically establish the randomness properties of the keystream [47]. RC4 consists of a table of all the $N = 2^n$ possible n-bit

words and two n-bit pointers. In original RC4 n is 8, and thus it has a huge state of $log_2(2^8! \times (2^8)^2) \approx 1700$ bits. It is thus impossible to guess even a small part of this state and almost all the techniques developed to attack stream ciphers based on linear feedback shift registers (LFSR) fail on RC4. Although state recovery attacks against RC4 have not been successful, several distinguishing attacks have been found on RC4. A few attempts have also been made to improve the design of RC4 against these attacks. Two significant ones are VMPC [118] and RC4A [98]. Both VMPC and RC4A have designs similar to RC4 and were shown to suffer from similar distinguishing attacks [112]. Recently some stream ciphers have been proposed that consist of a large internal state that changes under its own control just like RC4. However the state update functions in these ciphers are more complex and are designed to achieve a rapid change in state with time. Three ciphers that fall in this category are HC-256 [115], Py [9], and MV3 [60].

When RC4 was developed most commercially available processors were either 8-bit or 16-bit. Using the word size $n = 8$ was suitable for these processors and the amount of memory required to store the RC4 table was also feasible. Today the processors have word lengths of 32 bits or 64 bits but the most common mode for RC4 still uses $n = 8$. The reason is that for RC4 with $n = 32$ or $n = 64$, the size of the memory required to store the table and the key initialization time are too high. Still, since the modern processors have large word sizes it is of interest to design an RC4-like stream cipher that can take advantage of a large word size.

We begin our design by modifying original RC4 algorithm so that it can exploit the 32-bit and 64-bit processor architectures without increasing the size of the table significantly. We call the proposed algorithm RC4(n, m), where n and m are defined in Section 4.3. The algorithm is general enough to incorporate different word as well as table sizes. For example with 32-bit word size a table of 256 words can be used. We try to keep the original structure of RC4 as much as possible, however the proposed changes affect some underlying design principles on which the security of RC4 is based. Therefore we analyze the security of the modified RC4 and compare it to the original RC4. We show that RC4(n, m) is faster than RC4 and is also secure against several attacks that are successful on RC4.

The rest of the chapter is organized as follows. In Section 4.2 we give a brief description of original RC4. In Section 4.3 we propose a modified RC4 keystream generator. The security of the proposed generator is analyzed in Section 4.4 followed by a performance analysis in Section 4.5. Finally we present the known attacks on our proposed generator in Section 4.6 and possible countermeasures in Section 4.7.

4.2 Original RC4

In this section we give a description of the original RC4. We also give a brief description of previous attacks on RC4.

4.2.1 Description of RC4

The RC4 algorithm consists of two parts: The key scheduling algorithm (KSA) and the pseudo-random generation algorithm (PRGA). The algorithms are shown in Figure 4.1 where l is the length of the secret key in bytes, and N is the size of the array S or the S-box in words. A common keysize in RC4 is between 5 and 32 bytes. In most applications RC4 is used with a word size $n = 8$ and array size $N = 2^8$. In the first phase of RC4 operation an identity permutation $(0, 1, ..., N-1)$ is loaded in the array S. A secret key K is then used to initialize S to a random permutation by shuffling the words in S. During the second phase of the operation, the PRGA produces random words from the permutation in S. Each iteration of the PRGA loop produces one output word which constitutes the running keystream. The keystream is bit-wise XORed with the plaintext to obtain the ciphertext. All the operations described in Figure 4.1 are byte operations ($n = 8$). Most modern processors however operate on 32-bit or 64-bit words. If the word size in RC4 is increased to $n = 32$ or $n = 64$, to increase its performance, the size of array S becomes 2^{32} or 2^{64} bytes which is not practical. Note that these are the array sizes to store all the 32-bit or 64-bit permutations respectively.

```
KSA(K, S)

for  i = 0 to N – 1
  S[i] = i;
j = 0;
for  i = 0 to N – 1
  j = (j + S[i] + K[i mod l]) mod N;
  Swap(S[i],S[j]);
```

```
PRGA(S)

i = 0;
j = 0;
while (1)
  i = (i + 1) mod N;
  j = (j + S[i]) mod N;
  Swap(S[i],S[j]);
  out = S[(S[i] + S[j]) mod N];
```

Figure 4.1: The KSA and PRGA in RC4.

4.2.2 Previous analysis of RC4

Cryptanalysis of RC4 attracted a lot of attention in the cryptographic community after it was made public in 1994. Numerous significant weaknesses were discovered, including Finney's forbidden states [42], classes of weak keys [107], patterns that appear with twice the expected probability (the second byte bias) [69], partial message recovery [69], full key recovery attacks [44], analysis of biased distribution of RC4 initial permutation [77], and predicting and distinguishing attacks [68].

Knudsen et al. have attacked versions of RC4 with $n < 8$ by their backtracking algorithm [64]. The most serious weakness in RC4 was observed by Fluhrer et al. in [44] where RC4 was proved to have a practical attack in the security protocol WEP.

Two variants of RC4 have recently been proposed: RC4A [98] and VMPC [118]. RC4A works with two RC4 arrays and its keystream generation stage is slightly more efficient than RC4's, but the initialization stage requires twice the effort of RC4's. VMPC has several changes to the KSA, the IV integration, the round operation and the output selection. Note that RC4A and VMPC use $n = 8$ as parameter. Maximov described in [74] a linear distinguisher for both the variants, requiring 2^{58} data for RC4A and requiring 2^{54} data for VMPC. Tsunoo et al. described in [112] a distinguisher for RC4A and VMPC keystream generators, requiring 2^{24} and 2^{23} keystream prefixes respectively. For further weaknesses of RC4, and most of the

known attacks on it see [48, 42, 58, 47, 45, 78, 56, 107, 64, 69, 44, 110, 101, 77, 97, 98, 68, 8, 74, 112, 70].

4.3 Proposed Modification to RC4

We now propose a modification to the original RC4 algorithm which enables us to release 32 bits or 64 bits in each iteration of the PRGA loop. This is done by increasing the word size to 32 or 64 while keeping the array size S much smaller than 2^{32} or 2^{64}. We will denote the new algorithm as RC4(n, m) where $N = 2^n$ is the size of the array S in words, m is the word size in bits, $n \leq m$ and $M = 2^m$. For example RC4$(8, 32)$ means that the size of the array S is 256 and each element of S holds 32-bit words.

4.3.1 Pseudo-Random Generation Algorithm

If we choose n to be much smaller than m ($m = 32$ or 64) in RC4(n, m), then this results in reasonable memory requirements for the array S. However now the contents of the array S do not constitute a complete permutation of 32-bit or 64-bit words. In RC4, a swap operation is used to update the state between outputs. Using a swap to update the state in RC4(n, m) will not change the elements in the array. Instead, to update the state we add an integer addition modulo 2^{32} (2^{64} for $n = 64$). This way of updating the state is the first difference between RC4 and RC4(n, m). Since the state will be updated by replacing a random element by another random m-bit number, the swap operation is not needed. The index value that is updated is the value used for computing the output value. Updating the array with new values is important since the array is not a permutation and the size of the array is only a small fraction of all the possible numbers in $\mathbb{Z}_M$.

The second main difference between original RC4 and this variant is the usage of a third variable, k, in addition to i and j. This m-bit variable is used for two reasons. First, to mask the output so that it does not simply represent a value stored in the array. Second, to ensure that the new value in the update step does

```
KSA(K, S)

for  i = 0 to N – 1
 S[i] = a_i;
j = k = 0;
Repeat r times
 for  i = 0 to N – 1
  j = (j + S[i] + K[i mod l]) mod N;
  Swap(S[i],S[j]);
  S[i] = S[i] + S[j] mod M;
  k = k + S[i] mod M;
```

```
PRGA(S)

i = 0;
j = 0;
while (1)
 i = (i + 1) mod N;
 j = (j + S[i]) mod N;
 k = (k + S[j]) mod M;
 out = (S[(S[i] + S[j]) mod N] + k) mod M;
 S[(S[i] + S[j]) mod N] = k + S[i] mod M;
```

Figure 4.2: The modified KSA and PRGA for RC4(n, m).

not depend on just one or a few values in the array. The variable k is initialized in the KSA and is key dependent.

4.3.2 Key Scheduling Algorithm

The key scheduling algorithm (KSA) in RC4 is used to permute the elements in the array in a key dependent way. Each element is swapped with a random element. In this variant of RC4 the elements will not be a permutation of a small set so a similar modification is made to the KSA as to the PRGA. In order to achieve a high degree of randomness in the key scheduling we keep the swap operation in the KSA. In addition to the swap operation each word is updated through an integer addition. We give some initial values, a_i, for RC4$(8, 32)$, in Appendix B. The modified KSA and PRGA are given in Figure 4.2 where $N = 2^n$, $M = 2^m$, K is a vector of bytes and l is the length of the key K in bytes. RC4(n, m) can use the same flexible span of keysizes as RC4. The value of r in the KSA is motivated below and for a random array with 256 32-bit numbers the value of r is 20.

We take the example of 256, 32-bit numbers to motivate the number of steps used in the KSA. The array is initiated with 256 fixed 32-bit numbers and after the key scheduling algorithm the goal is that without knowing any bits of the

key an attacker can not guess the number in any array position with probability significantly greater than 2^{-32}. Since the array only contains a small fraction of all 32-bit numbers, the entries need to be updated. We update as the sum of the two swapped entries. After running through the array once, the probability that value i is not updated is

$$\left(\frac{255}{256}\right)^{256} \approx 0.37,$$

so a known value will be in the array with probability ≈ 0.37. The probability that this value is not updated after r rounds is 0.37^r. For a random array with 256 32-bit numbers the probability that a specific number is in the array is

$$1 - (1 - 2^{-32})^{256} \approx 2^{-24}$$

since a value can be present more than once in our case. We run the key initialization a sufficient number of rounds so that any initial value remains unupdated with probability $\leq 2^{-24}$. Hence, the number of rounds, r, we need in the initialization satisfies

$$\left(\frac{255}{256}\right)^{256r} = 2^{-24} \Rightarrow r \approx 16.6.$$

For the case RC4$(8, 32)$ we will take the value of r to be 20. Similarly the value of r can be calculated for different array sizes and different numbers of bits. In Table 4.1 we list the minimum number of rounds needed in the key scheduling such that no number has significantly higher probability of being in the array than any other number. We suggest to always use 20 rounds in the 32-bit version and always 40 rounds in the 64-bit version, when the array size is between 2^8 and 2^{12}.

4.4 Security Analysis of RC4(n, m)

In this section we analyze the security of RC4(n, m). We show that RC4(n, m) resists all known significant attacks on RC4. We consider the resistance of the generator against state recovery attacks and the randomness properties of the keystream.

Mode	r	Mode	r
RC4$(8, 32)$	16.6	RC4$(8, 64)$	38.7
RC4$(9, 32)$	15.9	RC4$(9, 64)$	38.1
RC4$(10, 32)$	15.2	RC4$(10, 64)$	37.4
RC4$(11, 32)$	14.6	RC4$(11, 64)$	36.7
RC4$(12, 32)$	13.9	RC4$(12, 64)$	36.1

Table 4.1: The minimum number of rounds in the key scheduling.

4.4.1 Statistical Tests on the Keystream

The keystream generated by the RC4$(8, 32)$ stream cipher was tested with NIST's statistical tests [93]. No bias was found by any of the 16 tests from the NIST suite. We tested 2^{35} output bits from the generator.

4.4.2 Security of the Key Scheduling Algorithm

We choose the number of steps in the key scheduling algorithm such that the probability that a specific number is not updated is smaller than the probability that this number is present in a random array of size N with m-bit numbers. This ensures that an attacker can not guess an array entry with probability significantly higher than $N/2^m$ when key generation starts. However, even if r is small it is unclear if an attacker can use the information about the values in the array in an actual attack. This is because the first output is the sum of $20 \cdot 256 + 2$ for RC4$(n, 32)$ and $40 \cdot 256 + 2$ for RC4$(n, 64)$ previous and current values in the array.

4.4.3 Internal State of RC4(n, m)

Like the original RC4, the security of RC4(n, m) comes from its huge internal state. The size of the internal state of original RC4 is approximately 1700 bits. In case of RC4(n, m) the internal state does not consist of a permutation and it may have repetitions of words. The number of ways of putting 2^m elements into N cells where repetitions are allowed is $(2^m)^N$. Note, in RC4(n, m) we are using an m-bit variable

k, which can be thought of as another cell. Therefore the size of the internal state is simply given by $N^2 \times (2^m)^{N+1}$. For example for RC4$(8, 32)$ this number is 8240 bits which is much larger than original RC4. Recovering the internal state of RC4(n, m) is therefore much harder than recovering the internal state of RC4.

4.4.4 Resistance to IV Weakness

Fluhrer, Mantin and Shamir showed in [44] a key recovery attack on RC4 if several IVs were known. The attack will work if the IV precedes or follows the key. In [70], Mantin showed that XORing the IV and the key also allows for a key recovery attack in the chosen IV model. The attack in which the IV precedes the key relies on the fact that the state at some point is in a *resolved* condition, which means that with probability 0.05, we can predict the output and also recover one byte of the key. Repeating the attack recovers another byte of the key etc. In RC4(n, m) this attack will not be possible. In the resolved condition the value i must be such that if $X = S_i[1]$ and $Y = S_i[X]$, then $i \geq 1$, $i \geq X$ and $i \geq X + Y$, where $S_i[V]$ is the entry $S[V]$ at time i. Moreover, $S_i[1]$, $S_i[X]$ and $S_i[X + Y]$ must be known to the attacker. Since the array is updated 20 times for RC4$(8, 32)$ and the key is used in all iterations, an attacker will not know the state after one iteration. Hence, the attacker can not know the state at a time when $i > 1$ in the last iteration which would be necessary for the attack to work. With similar arguments, we can conclude that the IV weakness in RC4 cannot be used for RC4(n, m) when the IV follows or is XORed with the key.

Concatenating the IV and the secret key does not seem to introduce an exploitable weakness to the cipher. However, we still consider it better to use a hash function on the secret key and IV and then use the hash value as session key. Then no related keys will be used if the IV is e.g., a counter. This is the mode used in SSL.

4.4.5 Resistance to Mantin's Distinguishing Attack

In [69] Mantin and Shamir discovered that the second output byte of RC4 is extremely biased, i.e., it takes the value of zero with probability $2/N$ instead of $1/N$. This is due to the fact that if $S_0[2] = 0$ and $S_0[1] \neq 2$, the second output byte of the keystream is zero with probability one. In RC4(n, m) the output is given by out $= (S[(S[i] + S[j]) \bmod N] + k) \bmod M$, where we assume that k is uniformly distributed. Therefore if $S_0[2] = 0$ and $S_0[1] \neq 2$ in RC4(n, m), the output word will still be uniformly distributed due to k. Therefore Mantin's distinguishing attack does not apply to RC4(n, m).

4.4.6 Resistance to Paul and Preneel's Attack

In [97] Paul and Preneel discovered a bias in the first two output bytes of the RC4 keystream. They observed that if $S_0[1] = 2$, then the first two output bytes of RC4 are always different. Therefore the probability that the first two output bytes are equal is $(1 - 1/N)/N$ which leads to a distinguishing attack. In RC4(n, m) however due to the uniform distribution of k, the above state does not affect the distribution of the first two output bytes. Therefore this attack does not apply to RC4(n, m).

4.4.7 Probability of Weak States

RC4 has a number of weak states, called Finney states [42]. These states have very short cycles, of length only 65280. The cipher is in a Finney state if $j = i + 1$ and $S[j] = 1$. In this case the swap will be made between $S[i]$ and $S[i + 1]$ and both i and j are incremented by 1. Since the RC4 next state function is an invertible mapping and the starting state is not a Finney state, RC4 will never enter any of these weak states. It is easy to see that RC4(n, m) also has weak states. When all entries are even and k is even, then all outputs as well as all future entries will be even, resulting in a biased keystream. The state update function in RC4(n, m) is not an invertible mapping so it will always be possible to enter one of these weak states. However the probability that all state entries, as well as k are even is very

low, 2^{-257}. From this we can conclude that these weak states are of no concern to the security of the cipher.

4.4.8 Forward Secrecy in RC4(n, m)

Like most of the keystream generators, RC4(n, m) keystream generator can also be represented as a finite state machine. Suppose $N = 2^n$, $M = 2^m$ and $R = \mathbb{Z}_N^2 \times \mathbb{Z}_M^{N+1}$. The next state function is $f : R \to R$. Let $(i, j, k, x_0, x_1, \cdots, x_{N-1}) \in R$ be any state, and $(e, d, p, y_0, y_1, \cdots, y_{N-1}) \in R$ be the next state of the function f. Then we have $e = i + 1 \bmod N$, $d = j + x_e \bmod N$, $p = k + x_d$, $v = x_e + x_d \bmod M$, $y_{v \bmod N} = k + x_e$ and $y_t = x_t$, $\forall t \neq v \bmod N$. Output of the cipher is $x_{v \bmod N} + p$. As seen above we can deterministically write down the value of each parameter of the next state. So given a state $(e, d, p, y_0, y_1, \cdots, y_{N-1})$, we can recover $(i, j, k, x_0, x_1, \cdots, x_{N-1})$ except x_v because x_v has been replaced. Therefore, without the knowledge of x_v the state function is non invertible.

4.4.9 Cycle Property

In original RC4 the state function is invertible. Non invertible state functions are known to cause a significantly shorter average cycle length. If the size of the internal state is s and the next state function is randomly chosen then the average cycle length is about $2^{\frac{s}{2}}$. For a randomly chosen invertible next state function the average cycle length is 2^{s-1}, (see [43]). As s in RC4$(8, 32)$ is huge (i.e., 8240) the reduction in cycle length is not a problem.

4.4.10 Randomness of the Keystream

To analyze the keystream of RC4(n, m) we first state the security principles underlying the design of original RC4. The KSA intends to turn an identity permutation S into a pseudorandom permutation of elements and PRGA generates one output byte from a pseudorandom location of S in every round. At every round the secret internal state S is changed by the swapping of elements, one in a known location

and another pointed to by a random index. Therefore we can say that the security of original RC4 depends on the following three factors.

- Uniform distribution of the initial permutation of elements in S.
- Uniform distribution of the value of index pointer j.
- Uniform distribution of the index pointer from which the output is taken (i.e., $(S[i] + S[j]) \bmod N$).

The above three conditions are necessary but not sufficient. The KSA uses a secret key to provide a uniformly distributed initial permutation of the elements in S. The value of the index pointer j is updated by the statement $j = (j + S[i]) \bmod N$. Since the elements in S are uniformly distributed the value of j is also uniformly distributed. By the same argument $(S[i] + S[j]) \bmod N$ is also uniformly distributed. Note that the internal state of RC4 consists of the contents of array S and the index pointer j. The state update function consists of an update of the value of j and the update of the permutation in S through a swap operation given by the statement $\text{Swap}(S[i], S[j])$. Since j is updated in a uniformly distributed way, the selection of the locations to be swapped is also uniformly distributed. This ensures that the internal state of RC4 evolves in a uniformly distributed way.

We now consider RC4(n, m). The first difference from original RC4 is that whereas the array S in original RC4 is a permutation of all the 256 elements in $\mathbb{Z}_{2^8}$, the array S in RC4(n, m) only contains 2^n m-bit words out of 2^m possible words in $\mathbb{Z}_{2^m}$. Consider the PRGA and assume that the initial permutation of 2^n elements in S is uniformly distributed over $\mathbb{Z}_{2^m}$. Then the index pointer j is updated by the statement

$$j = j + S[i] \mod N$$

where $j \in \mathbb{Z}_{2^n}$ and $S[i] \in \mathbb{Z}_{2^m}$. If the value of $S[i]$ is uniformly distributed over $\mathbb{Z}_{2^m}$, the value of index pointer j is also uniformly distributed over $\mathbb{Z}_{2^n}$. This implies that the value of the index pointer from which the output is taken (i.e., $S[i] + S[j] \bmod N$) is uniformly distributed over $\mathbb{Z}_{2^n}$. For the above properties to hold during PRGA phase it is essential that the internal state of the RC4(n, m) evolves in a

uniformly distributed manner. Recall that in original RC4 the uniform distribution of pointer j was the reason for the state to evolve uniformly since all the 256 elements in $\mathbb{Z}_{2^8}$ were present in the state. However in RC4(n, m) this is not the case and the uniform distribution of j over $\mathbb{Z}_{2^n}$ is not sufficient. The state update function also consists of the update of an element in S by integer addition modulo M given by the statement

$$S[S[i] \ + \ S[j] \mod N] = k \ + \ S[i] \mod M.$$

Since both k and $S[i]$ are uniformly distributed, the updated element in the state is also uniformly distributed. The internal state of RC4(n, m) evolves in a uniformly distributed manner and therefore the output of the cipher is also uniformly distributed, i.e., all the elements from $\mathbb{Z}_{2^m}$ occur with equal probability.

4.5 Performance of RC4(n, m)

RC4$(n, 32)$ has been designed to exploit the 32-bit architecture of the current processors. If n is chosen such that the corresponding memory requirements are reasonable, RC4$(n, 32)$ can give higher throughput than the original 8-bit RC4. We now compare the performance of RC4$(8, 32)$ with 8-bit RC4 and several other stream ciphers with similar designs. According to the NESSIE Performance Document [90], RC4 throughput is about 1.1 bits/cycle on a Pentium III processor. The throughput of eSTREAM submissions Py and HC-256, as reported by their designers, are 2.8 bits/cycle on a Pentium III and 1.9 bits/cycle on a Pentium 4 respectively. MV3 is the most recent design and its reported throughput is 1.67 bits/cycle on a Pentium IV. RC4$(8, 32)$ however achieves a throughput of approximately 3.4 bits/cycle on a Pentium IV. Therefore it is approximately 3.1 times faster than RC4 on 32-bit machines. It is also considerably faster than HC-256, Py and MV3 primarily due to its simple structure. We also implemented RC4$(8, 64)$ on a 64-bit Ultrasparc IV and found it to be approximately 6 times faster than the 8-bit RC4. This speedup is significant when large files are encrypted.

Though the keystream generation is faster than original RC4, the key scheduling

algorithm is slower. This is due to the importance of sufficient randomness in the initial state when keystream generation starts. In a situation where many small packets are encrypted with different keys/IVs, RC4 might still be faster due to its faster KSA.

4.6 Attack on RC4(n, m)

In Asiacrypt 2006 Paul and Preneel [100] presented a distinguishing attack on RC4$(8, 32)$. In their paper they referred to RC4(8,32) as GGNH. The attack concentrates only on the least significant bytes of the words in the table S. Let $S_t[i_t]$ denote eight least significant bits of the 32-bit word at location i in table S at time t and let $z_{t(l)}$ denote the l-th bit in the 32-bit keystream word produced at time t. If $S_t[i_t] = S_{t+1}[j_{t+1}]$ and $S_t[j_t] = S_{t+1}[i_{t+1}]$ then $z_{t+1(0)} = 0$. It is easy to see that if the above is true then we can write $z_{t+1} = 2(k + S_t[i_t]) \bmod 2^{32}$ which implies $z_{t+1}(0) = 0$. We denote this event by E. Since our table size is 2^8 and we consider the table entries to be uniformly distributed, $P[E] = 2^{-16}$ and $P[z_{t+1(0)} = 0] = 1/2(1 + 2^{-16})$ which is greater than 0.5. Therefore by observing the least significant bytes of the keystream words, the authors of [100] were able to construct a distinguisher which has data and time complexity of $O(2^{32.89})$.

4.7 Countermeasures and Future Work

The above distinguishing attack shows that while the operations in RC4(n, m) are mostly designed to break linearity over $\mathbb{Z}_2$, more operations are required to break the linearity over $\mathbb{Z}_{2^{32}}$ and $\mathbb{Z}_{2^8}$. This can be achieved through efficient bit-wise operations available on all modern processors. A good target for these operations is the update of the value k. The updated value of k must have a complex relation with its previous value. This can be obtained by adding one or more bitwise operations besides the addition modulo 2^{32}. Since k is used both in updating the table and also computing the keystream word z, this should provide extra security against the type of distinguishing attacks presented above. However the modified algorithm

has to be thoroughly analyzed for security. We intend to incorporate these changes into our design in future and propose a newer version of the RC4-like keystream generator.

It must be noted that the bitwise operations are very fast and the addition of two or three bitwise operations will not affect the performance of the RC4(n, m). We still expect it to be approximately 3 times faster than RC4 on 32-bit platforms. Note that RC4, HC-256, and MV3 have throughputs which are less than 2 bits/cycle. Although Py has a throughput of 2.8 bits/cycle, it was shown to be vulnerable to a distinguishing attack [99]. To address the weakness, the designers of Py proposed a new version of the cipher PyPy [10] with a reduced throughput of 1.4 bits/cycle which again turned out to be vulnerable to another distinguishing attack [100]. Therefore we believe that with current designs being slower than 2 bits/cycle there is considerable room for strengthening RC4$(8, 32)$ by adding more operations to its update function. Another area of improvement is the key scheduling of RC4(n, m). It is much slower than that of RC4 since the entries of the array must be sufficiently random before keystream generation starts. An improvement of the KSA would also be an interesting future research direction.

Chapter 5

Algebraic Immunity of Boolean Functions

We now turn our focus from the design of stream ciphers to the cryptographic properties of nonlinear transformations used in these ciphers. In this chapter we analyze the algebraic immunity of Boolean functions used in stream ciphers. In the next two chapters we will analyze the algebraic immunity and other cryptographic properties of the S-boxes used in block ciphers. We discussed the role of nonlinear transformations in the design of symmetric ciphers in Chapter 1. Many LFSR based stream ciphers use nonlinear Boolean functions to destroy the linearity of the LFSR(s) output. Many of these designs have been broken by algebraic attacks. Therefore from a designer's point of view it is very important to understand the algebraic properties of the nonlinear Boolean functions that make these designs vulnerable to algebraic attacks. In this chapter we analyze a popular and cryptographically significant class of nonlinear Boolean functions for their resistance to algebraic attacks. We develop techniques to compute the algebraic immunity of Boolean functions based on Power mappings and give meaningful bounds on the algebraic immunity of functions based on Inverse, Kasami, Niho and Dobbertin exponents. The techniques developed in this chapter also give insight into the relation between the polynomial form of Boolean functions and their algebraic immunity.

5.1 Background

The idea behind the algebraic attacks is to express the cipher as a system of multivariate equations whose solution gives the secret key. The complexity of the attack depends heavily on the degree of these equations. For complete details of an algebraic attack on an LFSR based stream cipher refer to Section 1.6. We begin by looking at a brief history of these attacks. The algebraic attacks on stream ciphers composed of LFSR(s) and a nonlinear combining function f were first proposed by Courtois and Meier in [23, 24]. The authors presented several scenarios under which low degree equations exist for ciphers using a combining function f with small number of inputs. These low degree equations are obtained by producing low degree multiples of f, i.e., by multiplying f with a low degree function g such that fg is of low degree. In [75] Meier, Pasalic and Carlet reduced the scenarios (given in [23, 24]) under which low degree equations can exist to two and showed that existence of low degree equations is equivalent to the existence of low degree annihilators of f or $f + 1$.

Krause and Armknecht extended algebraic attacks to combiners with memory in [3]. They proved that algebraic equations always exist for such combiners and also gave an upper bound on the degree of such equations in terms of their input size and memory. In [21] Courtois further extended these attacks to combiners with memory and several outputs and provided an upper bound on the degree of equations for such combiners in terms of the size of their input, output and memory. An improvement on the algebraic attacks called fast algebraic attacks was presented in [20]. These attacks have been further examined in [4, 57]. The first algebraic attack on a block cipher was discussed in [108]. In [25] Courtois and Pieprzyk showed that AES can be attacked by solving a system of quadratic equations. This is possible because the only nonlinear component in AES, i.e., the S-box, can be expressed as a system of quadratic Boolean equations.

Since the existence of low degree equations for simple combiners, combiners with memory, and even S-boxes is important for algebraic attacks, Armknecht combined the three cases in [5]. He showed that finding low degree equations for simple combiners, combiners with memory, and S-boxes can be reduced to the same

problem of finding low degree annihilators.

In other direction there is increasing interest in the construction of Boolean functions with highest algebraic immunity. Some constructions have been proposed [28,15,29] that can achieve maximum possible algebraic immunity $\lceil \frac{n}{2} \rceil$, where n is the number of inputs to the function. But the constructed function lacks certain cryptographic properties making it unsuitable to be used in a cryptosystem. In the absence of a suitable construction the other option is to pick a function and test it for low degree equations. However all current techniques for finding the low degree equations are based on exhaustive search algorithms which are impractical for larger values of n.

Most algorithms for finding the low degree equations are based on the theory of Boolean functions. Even, polynomial functions and S-boxes, i.e., functions and S-boxes designed over finite field $\mathbb{F}_{2^n}, n > 2$, are analyzed according to this theory. For example the filter function $f : \mathbb{F}_{2^{16}} \rightarrow \mathbb{F}_2$ used in the stream cipher SFINKS [14] is a component of the inverse mapping in $\mathbb{F}_{2^{16}}$. S-box used in AES [26] and stream cipher SNOW [38] consists of inverse mapping in $\mathbb{F}_{2^8}$. A power mapping from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^n}$ can be decomposed into n component functions, from $\mathbb{F}_{2^n}$ to $\mathbb{F}_2$, called monomial trace functions.

In this chapter we use the theory of polynomial functions to analyze the algebraic immunity of monomial trace function. This approach allows us to obtain meaningful results that are very difficult to obtain from the theory of Boolean functions. For example we derive upper bounds on the algebraic immunity of many monomial trace functions that are much lower than the optimal upper bound presented in the literature. We show that algebraic immunity of functions based on inverse, Kasami, Niho and Dobbertin exponents [34] decreases drastically as n increases. Moreover the existing algorithms to determine the algebraic immunity (and finding low degree equations) of functions are very slow and are not practical for $n > 25$. Our approach has no such limitation. The algebraic immunity of any monomial trace function can be obtained directly from the formula regardless of the value of n. Similarly the low degree equations required for the algebraic attack are also obtained directly from the formula.

5.2 Algebraic Immunity $\mathcal{AI}$

A Boolean function f is said to admit an *annihilating function* g, if $f * g = 0$. In [75] $\mathcal{AI}$ of f, denoted by $\mathcal{AI}(f)$, is defined as the minimum value of d such that f or $f + 1$ admits an annihilating function of degree d. Proposition 1 of [75] states that existence of relations of the form $f * g = h$, where g and h have degree at most d, means the existence of annihilating function g' of degree at most d (as $f^2 * g + f * h = f * g + f * h = f * (g + h) = 0$). Therefore if for f we can find a function g such that degree of $f * g$ is d then we can say that $\mathcal{AI}(f) \leq d$.

Fact 1. [23, Theorem 6.0.1] *Let f be any Boolean function with n inputs. Then there is a Boolean function $g \neq 0$ of degree at most $\lceil \frac{n}{2} \rceil$ such that $f * g$ is of degree at most $\lceil \frac{n}{2} \rceil$.*

Fact 1 shows that the upper bound on the $\mathcal{AI}$ of any Boolean function is $\lceil \frac{n}{2} \rceil$. To establish an upper bound on the $\mathcal{AI}$ of a polynomial function f we will try to find multipliers g such that the degree of $f * g$ is less than $\lceil \frac{n}{2} \rceil$.

5.3 Monomial Trace Functions and Power Mappings

Monomial trace functions are represented by a single trace term in polynomial form. There are several compelling reasons to study the $\mathcal{AI}$ of monomial trace functions. Any polynomial function can be expressed as a sum of monomial trace functions. Therefore, for a constant multiplier g, the $\mathcal{AI}$ of any function f is upper bounded by the maximum $\mathcal{AI}$ of any monomial trace function in its polynomial representation. This bound may not always be tight but in certain cases it can reveal the weakness of a function against algebraic attacks. Another important class of functions are the functions that can be represented as monomial trace functions. These functions can easily be constructed from a power mapping in a finite field. Therefore they can be implemented efficiently and are good candidates for hardware oriented stream ciphers. Power mappings can be represented as $F : x \rightarrow x^a$ in $\mathbb{F}_{2^n}$ and are classified

based on exponent a. These mappings can easily be decomposed into monomial trace functions:

Let $\{\alpha_0, ..., \alpha_{n-1}\}$ and $\{\beta_0, ..., \beta_{n-1}\}$ be dual bases of $\mathbb{F}_{2^n}$. Then an S-box based on power mapping $(F : x \rightarrow x^a)$ can be represented as $F(x) = \sum_{j=0}^{n-1} Tr_1^n(\beta_j x^{2^i t})\alpha_j$, $x \neq 0$ and its component functions can be represented as monomial trace functions of the form $f_j(x) = Tr_1^n(\beta_j x^{2^i t})$, where $a \in C_t$ [52, page 56]. It is conventional to represent monomial trace functions in the form $Tr_1^n(\beta x^t)$ where t is a coset leader of C_t. Note that we can write $a = 2^{-i}t$ for some i. So for any exponent a we can write the monomial trace function in standard form as

$$f(x) = Tr_1^n(\beta x^a) = Tr_1^n(\beta x^{2^{-i}t}) = Tr_1^n(\beta^{2^i} x^t),$$

since $Tr_1^n(x) = Tr_1^n(x^2)$. Next we look at some important power mappings.

5.4 Some Well Known Power Mappings

The discussion in this section is based on [34]. Some well known power mappings that have been studied for use in S-boxes are Inverse, Gold, Kasami, Welch and Niho (see also [46, 59, 35]). These mappings are cryptographically significant because of their high nonlinearity. Gold and Kasami power functions are maximally nonlinear (their nonlinearity is $2^{n-1} - 2^{\frac{n-1}{2}}$) for odd n. Welch and Niho mappings are conjectured to be maximally nonlinear for odd n. For even n it is conjectured that for all power mappings, maximum achievable nonlinearity is $2^{n-1} - 2^{\frac{n}{2}}$. If a mapping achieves this nonlinearity it is called *maximally nonlinear*. For example inverse mapping is maximally nonlinear for even n. Note that monomial trace functions with the above mentioned exponents are simply the component functions of these mappings. Therefore the nonlinearity of these monomial trace functions is greater than or equal to the nonlinearity of their corresponding power mappings. Moreover, easy implementation and analysis of these monomial trace functions make them suitable candidates for combining or filtering functions in stream ciphers [14].

5.5 Algebraic Immunity of Monomial Trace Functions

We provide the following proposition that will be used to derive upper bounds on the $\mathcal{AI}$ of monomial trace functions.

Proposition 1 *Let $f(x) = Tr_1^n(\beta x^t)$ and $g(x) = Tr_1^m(\gamma x^r)$ be monomial trace functions, where $x \in \mathbb{F}_{2^n}$, t and r are the coset leaders of cosets C_t and C_r. The sizes of the cosets C_t and C_r are n and m respectively, $m|n$, and $\beta \in \mathbb{F}_{2^n}$, $\gamma \in \mathbb{F}_{2^m}$. Then*

$$deg(f(x)g(x)) = \max_{0 \le i < m} H(r + t2^{-i})$$

Proof. Note both $f(x)$ and $g(x)$ are n variable Boolean functions. From the definition of trace function we can write (see also [51])

$$f(x)g(x) = \sum_{j=0}^{n-1} (\beta x^t)^{2^j} \sum_{l=0}^{m-1} (\gamma x^r)^{2^l} = \sum_{j=0}^{n-1} \sum_{l=0}^{m-1} \beta^{2^j} \gamma^{2^l} x^{t2^j + r2^l}$$

$$= \sum_{k=0}^{m-1} Tr_1^n(\gamma \beta^{2^k} x^{r + t2^k}),$$

where the algebraic degree of $f(x)g(x)$ is given by the largest w such that $\gamma\beta^{2^k} \neq 0$ and $H(r + t2^k) = w$. Let $k = m - i$, we have $t2^k \equiv t2^{m-i} \equiv t2^{-i} \bmod 2^n - 1$. Therefore

$$deg(f(x)g(x)) = \max_{0 \le k < m} H(r + t2^k) = \max_{0 \le i < m} H(r + t2^{-i})$$

□

Proposition 1 shows that we only need to add r to the members of coset C_t, and the highest Hamming weight of the resulting integers gives the maximum possible degree of $f(x)g(x)$.

In the following theorem we derive an upper bound on the $\mathcal{AI}$ of monomial trace functions based on a property of the exponent t, i.e., the number of runs of

1's in the binary representation of t.

Theorem 1 *Let $l = \lfloor \sqrt{n} \rfloor$, $k = n - \lfloor \frac{n}{l} \rfloor l$ and $f(x) = Tr_1^n(\beta x^t)$, where $\beta \in \mathbb{F}_{2^n}$ and t is the coset leader of C_t. Let $g(x) = Tr_1^m(x^r)$, where*

$$m = \begin{cases} l, & k = 0; \\ n, & 0 < k < l \end{cases}, \text{ and } r = \begin{cases} 1 + \sum_{i=1}^{\frac{n}{l}-1} 2^{il} & , k = 0 \\ 1 + 2^k + \sum_{i=1}^{\lfloor \frac{n}{l} \rfloor - 1} 2^{il+k} & , 0 < k < l \end{cases}.$$

Then

$$deg(f(x)g(x)) \leq ul + \left\lceil \frac{n}{l} \right\rceil - 1, \tag{5.1}$$

where u is the number of runs of 1s in the binary representation of t.

To prove Theorem 1 we need the following two lemmas.

Lemma 1 *r is a coset leader modulo $2^n - 1$.*

Proof. The above can be established by examining the binary representation of r.

Case: $k = 0$

$$\begin{array}{lcl} & & \overbrace{\qquad\qquad\qquad\qquad\qquad\qquad}^{n} \\ & & \overbrace{00\cdots01}^{l}\overbrace{00\cdots01}^{l}\ldots\ldots\overbrace{00\cdots01}^{l} \\ r & = & 00\cdots01\,00\cdots01\ldots\ldots00\cdots01 \\ 2r & = & 00\cdots10\,00\cdots10\ldots\ldots00\cdots10 \quad > \quad r \\ \cdot & \cdot & \qquad\qquad\qquad\cdot \\ 2^{l-1}r & = & 10\cdots00\,10\cdots00\ldots\ldots10\cdots00 \quad > \quad r \\ 2^l r & = & 00\cdots01\,00\cdots01\ldots\ldots00\cdots01 \quad = \quad r \end{array}$$

Therefore r is the coset leader modulo $2^n - 1$.

Case: $0 < k < l$

$$
\begin{array}{lcl}
 & \overbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}^{n} & \\
 & \overbrace{00\cdots01}^{l}\overbrace{00\cdots01}^{l}\ldots\ldots\overbrace{00\cdots01}^{l}\overbrace{00\cdots01}^{k} & \\
r = & 00\cdots10\,00\cdots10\ldots\ldots00\cdots10\,00\cdots10 & > \; r \\
2r = & \cdot \qquad\qquad\qquad\qquad\cdot & \\
\cdot \quad \cdot & & \\
 & \overbrace{00\cdots01}^{l}\overbrace{00\cdots01}^{l}\ldots\ldots\overbrace{00\cdots01}^{k}\overbrace{00\cdots01}^{l} & \\
2^l r = & 00\cdots01\,00\cdots01\ldots\ldots00\cdots01\,00\cdots01 & > \; r \\
\cdot \quad \cdot & \cdot \qquad\qquad\qquad\qquad\cdot & \\
 & \overbrace{00\cdots01}^{l}\overbrace{00\cdots01}^{l}\ldots\ldots\overbrace{00\cdots01}^{l}\overbrace{00\cdots01}^{k} & \\
2^n r = & 00\cdots01\,00\cdots01\ldots\ldots00\cdots01\,00\cdots01 & = \; r
\end{array}
$$

Therefore r is the coset leader modulo $2^n - 1$. □

Note: For $k = 0, |C_r| = l$ and for $0 < k < l, |C_r| = n$, where $|C_r|$ is the size of the coset C_r.

In Lemma 1 $\overbrace{\quad}$ is used to indicate the size of a segment in bits. From here onwards we will use $\overbrace{\quad}$ to represent the size of the segment as before and $\underbrace{\quad}$ to represent the number of 1's in the segment.

Lemma 2 *$H(r + t2^{-i}) \leq ul + \lceil \frac{n}{l} \rceil - 1$, $0 \leq i \leq m - 1$ and r, t, u, and l are as defined in Theorem 1.*

Proof. Consider the binary representations of r and t. In Lemma 1, r consists of $\lfloor \frac{n}{l} \rfloor$ identical l bit segments when $k = 0$. If $k \neq 0$, then k least significant bits of r form an additional segment. All $\lceil \frac{n}{l} \rceil$ segments have Hamming weight 1. We can segment t in the same way as r however these segments may or may not be identical. We will represent a segment of r and t as r' and t' respectively. Now let us consider the addition of r' and t'. Initially we will restrict ourselves to the case where the binary representation of t' has at most one run. Now consider all possible transitions in t' with and without carry.

Case 1: 1 → 0 transition

$$\begin{array}{rcl} & & \overbrace{}^{j\ \curvearrowright} \\ t' &=& 111\cdots10\cdots00 \\ r' &=& 000\cdots00\cdots01 \\ &+& --------- \\ & & 111\cdots10\cdots01 \end{array} \qquad \begin{array}{rcl} & & \overbrace{}^{j\ \curvearrowright} \quad \leftarrow carry \\ t' &=& 111\cdots10\cdots00 \\ r' &=& 000\cdots00\cdots01 \\ &+& --------- \\ & & 111\cdots10\cdots10 \end{array}$$

$$H(r'+t') = j+1, j < l-1 \qquad H(r'+t') = j+1, j < l-1$$

$$\begin{array}{rcl} & & \overbrace{}^{j\ \curvearrowright} \\ t' &=& 111\cdots110 \\ r' &=& 000\cdots001 \\ &+& -------- \\ & & 111\cdots111 \end{array} \qquad \begin{array}{rcl} carry \leftarrow & & \overbrace{}^{j\ \curvearrowright} \quad \leftarrow carry \\ t' &=& 111\cdots110 \\ r' &=& 000\cdots001 \\ &+& -------- \\ & & 000\cdots000 \end{array}$$

$$H(r'+t') = j+1, j = l-1 \qquad H(r'+t') = 0, j = l-1$$

Case 2: 0 → 1 transition

$$\begin{array}{rcl} & & \overbrace{}^{j\ \curvearrowright} \\ t' &=& 000\cdots01\cdots11 \\ r' &=& 000\cdots00\cdots01 \\ &+& --------- \\ & & 000\cdots10\cdots00 \end{array} \qquad \begin{array}{rcl} & & \overbrace{}^{j\ \curvearrowright} \quad \leftarrow carry \\ t' &=& 000\cdots01\cdots11 \\ r' &=& 000\cdots00\cdots01 \\ &+& --------- \\ & & 000\cdots10\cdots01 \end{array}$$

$$H(r'+t') = 1, j < l \qquad H(r'+t') = 2, j < l$$

Case 3: No transition

$$\begin{array}{rcl} & & \\ t' &=& 000\cdots00 \\ r' &=& 000\cdots01 \\ &+& ------ \\ & & 000\cdots01 \end{array} \qquad \begin{array}{rcl} & & \leftarrow carry \\ t' &=& 000\cdots00 \\ r' &=& 000\cdots01 \\ &+& ------ \\ & & 000\cdots10 \end{array}$$

$H(r'+t')=1$ $\qquad\qquad$ $H(r'+t')=1$

$$\begin{array}{lcl} carry & \leftarrow & \\ t' & = & 111\cdots11 \\ r' & = & 000\cdots01 \\ & + & ------ \\ & & 000\cdots00 \end{array} \qquad \begin{array}{lcl} carry & \leftarrow & \leftarrow carry \\ t' & = & 111\cdots11 \\ r' & = & 000\cdots01 \\ & + & ------ \\ & & 000\cdots01 \end{array}$$

$H(r'+t')=0$ $\qquad\qquad$ $H(r'+t')=1$

From the above cases it is clear that $H(r'+t')$ achieves maximum value l for $1 \rightarrow 0$ transition (no carry case). For $0 \rightarrow 1$ transition the maximum value is 2 and when there is no transition it is 1. Now consider the following complete binary representation of r and t.

$$\begin{array}{rcl} & & \overbrace{\qquad}^{l} \qquad\qquad \overbrace{\qquad}^{l} \qquad\qquad \overbrace{\qquad}^{l} \\ t & = & \overbrace{**\cdots**}\ldots\ldots\overbrace{1110\cdots00}\ldots\ldots\overbrace{**\cdots**} \\ r & = & 0\,0\cdots0\,1\ldots\ldots0000\cdots01\ldots\ldots0\,0\cdots0\,1 \end{array}$$

where * can be either 1 or 0.

First we assume that each segment t' has at most one run. To get the maximum possible value of $H(r+t2^{-i})$ each segment with a $1 \rightarrow 0$ transition must contribute l number of 1's to the sum. Since there are u number of $1 \rightarrow 0$ transitions this adds up to ul number of 1's. For u number of $1 \rightarrow 0$ transitions there are at most u segments with $0 \rightarrow 1$ transitions, each contributing a maximum of 2 number of 1's to the sum, i.e., $2u$ number of 1's. All the remaining $\lceil \frac{n}{l} \rceil - 2u$ segments with no transitions can contribute at most single 1 to the sum. So the total contribution is $ul + 2u + \lceil \frac{n}{l} \rceil - 2u = ul + \lceil \frac{n}{l} \rceil$. Now consider the first segment from right to left that contains a $0 \rightarrow 1$ transition (right most segment with a $0 \rightarrow 1$ transition). For this segment to contribute 2 number of 1's to the sum it must receive a carry, otherwise it will contribute a single 1 (see case 2). Since the right most segment never receives a carry, the only way a carry can be generated between the right most segment and the right most segment with a $0 \rightarrow 1$ transition is due to the presence of a segment with all 1's (see all cases without carry). However this all 1's

segment contributes zero number of 1's to the sum. Therefore we subtract 1 from $ul + \lceil \frac{n}{l} \rceil$. Therefore the maximum possible value of $H(r + t2^{-i})$ is $ul + \lceil \frac{n}{l} \rceil - 1$.

Suppose a segment t' has more than one runs. Then it must contain a $1 \rightarrow 0$ transition. In our analysis of segments with single runs, we assumed that each segment with a $1 \rightarrow 0$ transition must contribute l number of 1's to the sum. As the size of the segment is l the contribution of a segment with more than one runs must be less than or equal to the contribution of the segment with one run. Therefore $H(r + t2^{-i})$ is upper bounded by $ul + \lceil \frac{n}{l} \rceil - 1$.

□

Proof of Theorem 1.
The assertion follows directly from Lemma 1, Lemma 2 and Proposition 1. □

5.5.1 Impact of Theorem 1 on Algebraic and Fast Algebraic Attacks

From the point of view of algebraic attacks on LFSR based stream ciphers, Theorem 1 is particularly significant. It not only gives the upper bound on the $\mathcal{AI}$ of f, i.e., $ul + \lceil \frac{n}{l} \rceil - 1$ but at the same time also gives the low degree multiplier g. In Theorem 1 we give only one multiplier however we can get $2^m - 1$ distinct non zero multipliers by taking $g(x) = Tr_1^m(\beta x^r)$, where $\beta \in \mathbb{F}_{2^m}$. Note only m of them are linearly independent.

An improvement on the conventional algebraic attacks is the fast algebraic attacks introduced by Courtois in [20] and later improved in [4, 57]. Functions with high $\mathcal{AI}$ may fail to provide resistance against fast algebraic attacks if the degree of the multiplier g is very low (see [2, 13, 27]). These attacks improve on conventional algebraic attacks by adding a pre-computation step which reduces the complexity of the online step of the attack. Usually the smaller the degree of g, the greater is the improvement. From Theorem 1 degree of g is $\lceil \frac{n}{\lfloor \sqrt{n} \rfloor} \rceil$, which is sufficiently small. Therefore an LFSR based stream cipher using nonlinear function f is vulnerable to fast algebraic attacks.

The first step in both algebraic and fast algebraic attacks is to find low degree

relations. The complexity of this step is roughly $D^2, D = \sum_{i=0}^{d} \binom{n}{d}$, where d is the $\mathcal{AI}$ of f [2]. For large values of n (say > 30) and d (say > 15) this computation may be infeasible. However Theorem 1 immediately gives several low degree relations for many classes of monomial trace functions regardless of the value of n.

Remark 1 *From Fact 1 we know that $\mathcal{AI}$ of any function is at most $\lceil \frac{n}{2} \rceil$. Let v be the degree of $f(x)$, then, to obtain a meaningful upper bound on $\mathcal{AI}$, $deg(f(x)g(x)) \leq \min\left(v, \lceil \frac{n}{2} \rceil\right)$. So we have the following condition on u,*

$$u \leq \min\left(\frac{v - \lceil \frac{n}{l} \rceil + 1}{l}, \frac{\lceil \frac{n}{2} \rceil - \lceil \frac{n}{l} \rceil + 1}{l}\right). \tag{5.2}$$

For many cryptogaphically useful power mappings, u is very small. For example, $u = 1$ for inverse, and $u = 2$ for Kasami, Gold, Welch and Niho. Therefore Theorem 1 can give very useful bounds for these mappings. In fact in most cases using the proof technique of Theorem 1 and exploiting the specific binary form of each exponent, we can further improve this bound. Since functions with Gold and Welch exponents have very small degrees (2 and 3 respectively) we will only consider inverse, Kasami, Niho and Dobbertin exponents in this chapter.

5.6 Inverse Exponent

Inverse mappings $x \rightarrow x^{-1}$ in $\mathbb{F}_{2^n}$ can be decomposed in n monomial trace functions of the form $Tr_1^n(\beta x^{-1})$. The degree of these monomial trace functions is $n-1$. Among all bijective Boolean functions, degree $n-1$ is only achieved by the monomial trace functions with inverse exponent. The inverse exponent consists of a single run of 1's. From Theorem 1 its $\mathcal{AI}$ is upper bounded by $l + \lceil \frac{n}{l} \rceil - 1$. However in Theorem 2 we show that for inverse function this bound can be improved to $l + \lceil \frac{n}{l} \rceil - 2$.

Lemma 3 *Let $t = 2^{n-1} - 1$. Then $H(r + t2^{-i}) \leq l + \lceil \frac{n}{l} \rceil - 2$, $0 \leq i \leq m-1$ and r is defined as in Theorem 1.*

Proof. Consider the binary representation of r and t.

$$\begin{array}{rcl} & & \overbrace{}^{l}\overbrace{}^{} \cdots\cdots \overbrace{}^{l}\overbrace{}^{l} \\ t & = & 01\cdots11\,11\cdots11\cdots\cdots11\cdots11\,11\cdots11 \\ r & = & 00\cdots01\,00\cdots01\cdots\cdots00\cdots01\,00\cdots01 \\ & + & ----------\text{-}---- \\ & & \underbrace{10\cdots01}_{2}\underbrace{00\cdots01}_{1}\cdots\cdots\underbrace{00\cdots01}_{1}\underbrace{00\cdots00}_{0} \end{array}$$

$H(r+t) = \lceil \frac{n}{l} \rceil$. We can see that $H(r + t2^{-i})$ is maximized when $i = l - 2$

$$\begin{array}{rcl} & & \overbrace{}^{l}\overbrace{}^{} \cdots\cdots \overbrace{}^{l}\overbrace{}^{l} \\ t2^{-(l-2)} & = & 11\cdots01\,11\cdots11\cdots\cdots11\cdots11\,11\cdots11 \\ r & = & 00\cdots01\,00\cdots01\cdots\cdots00\cdots01\,00\cdots01 \\ & + & ----------\text{-}---- \\ & & \underbrace{11\cdots11}_{l}\underbrace{00\cdots01}_{1}\cdots\cdots\underbrace{00\cdots01}_{1}\underbrace{00\cdots00}_{0} \end{array}$$

$H(r + t2^{-(l-2)}) = l + \lceil \frac{n}{l} \rceil - 2$. Therefore $H(r + t2^{-i}) \leq l + \lceil \frac{n}{l} \rceil - 2$, $0 \leq i \leq n-1$. □

Theorem 2 *Let* $f(x) = Tr_1^n(\beta x^{-1})$ *and* $g(x) = Tr_1^m(x^r)$. *Then*

$$deg(f(x)g(x)) \leq l + \left\lceil \frac{n}{l} \right\rceil - 2 \tag{5.3}$$

where β, m, r *and* l *are the same as defined in Theorem 1.*

Proof. From Lemma 3, $t = 2^{n-1} - 1$. Since $Tr_1^n(x) = Tr_1^n(x^2)$, we have

$$f(x) = Tr_1^n(\beta x^{-1}) = Tr_1^n(\beta x^{2t}) = Tr_1^n(\beta^{2^{n-1}} x^t). \tag{5.4}$$

From Lemma 1, r is a coset leader and from Lemma 3, Eqn.(5.4) and Proposition 1, $deg(f(x)g(x)) \leq l + \lceil \frac{n}{l} \rceil - 2$. □

In their Eurocrypt 2006 paper [2] Armknecht et al. experimentally determined the exact $\mathcal{AI}$ of the monomial trace functions with inverse exponents for $12 \leq n \leq 20$. Their results are the same as the upper bound given in Theorem 2. This shows that our bound is tight for these values of n. We believe that this bound is (very likely) tight for all values of n. Also note that this bound on $\mathcal{AI}$ is much less than theoretical optimal value $\lceil \frac{n}{2} \rceil$ for higher values of n (see Table 5.1).

5.7 Kasami and Kasami-Like Exponents

A Kasami exponent [59], e, is defined as $e = 2^{2s} - 2^s + 1$, $\gcd(n, s)$=1 and $1 \leq s \leq \frac{n}{2}$. If we remove the condition $\gcd(n, s)$=1 we can write $\acute{e} = 2^{2s} - 2^s + 1$, $1 \leq s \leq \frac{n}{2}$. We will call $\acute{e}$, the Kasami-like exponent. The motivation behind studying Kasami-like exponents, instead of only Kasami exponents, comes from [2]. The authors of [2] experimentally determined the $\mathcal{AI}$ of Kasami and Kasami-like exponents for $12 \leq n \leq 20$. Based on their results, they suggested that such functions have high $\mathcal{AI}$. However we show that monomial trace functions with Kasami or Kasami-like exponents have poor $\mathcal{AI}$ for larger values of n. Also by considering Kasami-like exponents we get a much larger class of functions with $\mathcal{AI}$ upper bounds the same as Kasami exponents.

An n variable monomial trace function with a Kasami-like exponent, $f(x)$, can be defined as $f(x) = Tr_1^n(\beta x^{\acute{e}})$. The algebraic degree of $f(x)$ is $s+1$. The Kasami-like exponents consist of 2 runs of 1's in their binary representation. From Theorem 1 their $\mathcal{AI}$ is upper bounded by $2l + \lceil \frac{n}{l} \rceil - 1$. However in Theorem 3 we show that this bound can be improved further.

Lemma 4 *Let $t = \acute{e}2^{-s}$ where $\acute{e} = 2^{2s} - 2^s + 1$, $1 \leq s \leq \frac{n}{2}$. Then*

$$H(r + t2^{-i}) \leq \begin{cases} l + \lceil \frac{n}{l} \rceil, & s \not\equiv 1 \ mod\ l \\ l + \lceil \frac{n}{l} \rceil - 1, & s \equiv 1 \ mod\ l \end{cases}$$

where $0 \leq i \leq m-1$ and r is defined in Theorem 1.

Proof. The binary representation of $\acute{e}$ is:

$$\acute{e} \;=\; 2^{2s} - 2^s + 1 \;= \overbrace{00\cdots00}^{n-2s}\overbrace{11\cdots11}^{s}\overbrace{00\cdots01}^{s}.$$

Let $t = \acute{e}2^{-s}$, then

$$t \;=\; 2^{2s} - 2^s + 1 \;= \overbrace{00\cdots01}^{s}\overbrace{00\cdots00}^{n-2s}\overbrace{11\cdots11}^{s}.$$

Consider the addition of r and t.

Case 1: $s \not\equiv 1 \bmod l$

The binary representation of t has 2 runs of 1's out of which one run consists of single 1. This 1 can only contribute a single 1 in any segment of $r + t2^{-i}$. Now we can see that $H(r + t2^{-i})$ is maximized when $i = l - 1$.

$$\begin{array}{lcl}
t2^{-(l-1)} & = & \overbrace{11\cdots10}^{l}\overbrace{00\cdots00}\ldots\ldots\overbrace{0\cdots100}^{l}\overbrace{001\cdots11}^{l}\ldots\ldots\overbrace{11\cdots11} \\
r & = & 00\cdots01\,00\cdots01\ldots\ldots0\cdots001\,000\cdots01\ldots\ldots00\cdots01 \\
 & + & ----------\text{-}----------\text{-}---------- \\
 & & \underbrace{11\cdots11}_{l}\underbrace{00\cdots01}_{1}\ldots\ldots\underbrace{0\cdots101}_{2}\underbrace{010\cdots01}_{2}\ldots\ldots\underbrace{00\cdots00}_{0}
\end{array}$$

$H(r + t2^{-(l-1)}) = l + \lceil \frac{n}{l} \rceil$.

Case 2: $s \equiv 1 \bmod l$

When $i = l-1$, the run with a single 1, in the binary representation of $t2^{-i}$, is always at the least significant position of a segment. Therefore it does not contribute an additional 1 in the corresponding segment of $r + t2^{-i}$.

$$
\begin{array}{lcl}
 & & \overbrace{}^{l}\overbrace{} \cdots\cdots \overbrace{}^{l}\overbrace{}^{l} \cdots\cdots \overbrace{} \\
t2^{-(l-1)} & = & 11\cdots10\,00\cdots00 \cdots\cdots 0\cdots001\,001\cdots11 \cdots\cdots 11\cdots11 \\
r & = & 00\cdots01\,00\cdots01 \cdots\cdots 0\cdots001\,000\cdots01 \cdots\cdots 00\cdots01 \\
 & + & ---------- \\
 & & \underbrace{11\cdots11}_{l}\underbrace{00\cdots01}_{1} \cdots\cdots \underbrace{0\cdots010}_{1}\underbrace{010\cdots01}_{2} \cdots\cdots \underbrace{00\cdots00}_{0}
\end{array}
$$

$H(r + t2^{-(l-1)}) = l + \lceil \frac{n}{l} \rceil$-1. □

Theorem 3 *Let $f(x) = Tr_1^n(\beta x^{\acute{e}})$, where é is a Kasami-like exponent. Then*

$$deg(f(x)g(x)) \leq \begin{cases} l + \left\lceil \frac{n}{l} \right\rceil, & s \not\equiv 1 \ mod\ l \\ l + \left\lceil \frac{n}{l} \right\rceil - 1, & s \equiv 1 \ mod\ l \end{cases}$$

where β, l and $g(x)$ are defined in Theorem 1.

Proof. Let $t = \acute{e}2^{-s}$, then

$$f(x) = Tr_1^n(\beta x^{\acute{e}}) = Tr_1^n(\beta x^{t2^s}) = Tr_1^n(\beta^{2^{n-s}} x^t), \tag{5.5}$$

since $Tr_1^n(x) = Tr_1^n(x^2)$. The assertion follows directly from Lemma 4, Lemma 1, Eqn.(5.5) and Proposition 1. □

In [2] the exact $\mathcal{AI}$ of the monomial trace functions with Kasami-like exponents for $12 \leq n \leq 20$ is determined experimentally. For $n = 19$ the results agree with our bound. For the remaining values of n the results are very close to our bound.

Remark 2 *In Theorem 3 bound on $\mathcal{AI}$ is proved for Kasami-like exponents. Since Kasami exponents are a subset of Kasami-like exponents these bounds also hold for Kasami exponents. Note that these bounds are much lower than the optimal bound $\lceil \frac{n}{2} \rceil$ for large n (see Table 5.1).*

Remark 3 *For cryptographic applications it is desirable for a Boolean function to have high degree. Let $f(x) = Tr_1^n(\beta x^e)$ be a maximum degree function such that e*

is a Kasami exponent. If n is odd then degree of $f(x)$ is $\frac{n+1}{2}$ and it is easy to see that the binary representation of e consists of one run with a single 1 followed by one run with a single 0. Only one of this run can contribute an additional 1 in a segment of $r + t2^{-i}$. Therefore the $\mathcal{AI}$ of an odd variable, highest degree monomial trace function with Kasami exponent is upper bounded by $l + \lceil \frac{n}{l} \rceil - 1$.

5.8 Niho and Dobbertin Exponents

First we consider Niho exponent. An n variable monomial trace function with Niho exponent [35,91], $f(x)$, can be defined as $f(x) = Tr_1^n(\beta x^e)$, where $e = 2^s + 2^{\frac{s}{2}} - 1$, $n = 2s + 1$ when s is even and $e = 2^s + 2^{\frac{3s+1}{2}} - 1$, $n = 2s + 1$ when s is odd. The degree of Niho function in n variables is $\frac{n+3}{4}$ for $n \equiv 1 \bmod 4$ and $\frac{n+1}{2}$ for $n \equiv 3 \bmod 4$.

The Niho exponent consists of 2 runs of 1's in its binary representation. From Theorem 1 its $\mathcal{AI}$ is upper bounded by $2l + \lceil \frac{n}{l} \rceil - 1$. However we improve this bound in Theorem 4.

Lemma 5 *Let $t = e$ be a Niho's exponent. Then*

$$H(r + t2^{-i}) \leq \begin{cases} l + \lceil \frac{n}{l} \rceil - 1, & s \equiv 0 \ mod\ l,\ s \ is\ even\ or \\ & s \equiv 1 \ mod\ l,\ s \ is\ odd \\ l + \lceil \frac{n}{l} \rceil, & otherwise \end{cases}$$

where $0 \leq i \leq m - 1$ and r is defined in Theorem 1.

Proof. The binary representation of t is:

$$t = 2^s + 2^{\frac{s}{2}} - 1 = \overbrace{00\cdots01}^{s+1}\overbrace{00\cdots00}^{\frac{s}{2}}\overbrace{11\cdots11}^{\frac{s}{2}},\ n = 2s + 1,\ \text{s is even}$$

$$t = 2^s + 2^{\frac{s}{2}} - 1 = \overbrace{00\cdots01}^{\frac{s+1}{2}}\overbrace{00\cdots00}^{\frac{s+1}{2}}\overbrace{11\cdots11}^{s},\ n = 2s + 1,\ \text{s is odd}$$

The binary representation of t has 2 runs of 1's out of which one run consists of single 1. This 1 can only contribute a single 1 in any segment of $r+t$. As in Lemma 4 it is easy to verify that $H(r+t2^{-i})$ is maximized when $i=l-1$. If $s \equiv 0 \bmod l$ and s is even or $s \equiv 1 \bmod l$ and s is odd, then the run with a single 1, in the binary representation of $t2^{-i}$, is always at the least significant position of a segment. Therefore it does not contribute an additional 1 in the corresponding segment of $r+t$ (see Lemma 4). So $H(r+t2^{-(l-1)})$ is given by $l+\lceil \frac{n}{l} \rceil - 1$.

In all the other cases this 1 can contribute an additional 1 in the corresponding segment of $r+t$, and therefore in such cases $H(r+t2^{-(l-1)})$ is given by $l+\lceil \frac{n}{l} \rceil$. □

Theorem 4 *Let $f(x) = Tr_1^n(\beta x^e)$, where e is a Niho exponent. Then*

$$deg(f(x)g(x)) \leq \begin{cases} l+\lceil \frac{n}{l} \rceil - 1, & s \equiv 0 \ mod\ l,\ s \ is\ even\ or \\ & s \equiv 1 \ mod\ l,\ s \ is\ odd \\ l+\lceil \frac{n}{l} \rceil, & otherwise \end{cases}$$

where β, l and $g(x)$ are defined in Theorem 1.

Proof. The assertion follows directly from Lemma 5, Lemma 1, and Proposition 1. □

Now we consider Dobbertin exponent. An n variable monomial trace function with Dobbertin exponent [35], $f(x)$, can be defined as $f(x) = Tr_1^n(\beta x^e)$, where $e = 2^{4s}+2^{3s}+2^{2s}+2^s-1$ and $n=5s$. The degree of $f(x)$ is $s+3$. The binary representation of e has 4 runs of 1's. From Theorem 1 its $\mathcal{AI}$ is upper bounded by $4l+\lceil \frac{n}{l} \rceil - 1$. However in Theorem 5 we show that $\mathcal{AI}$ of $f(x)$ is upper bounded by $l+\lceil \frac{n}{l} \rceil + 2$.

Theorem 5 *Let $f(x) = Tr_1^n(\beta x^e)$,where e is a Dobbertin exponent. Then*

$$deg(f(x)g(x) \leq\ l+\lceil \frac{n}{l} \rceil + 2$$

where β, l, and $g(x)$ are defined in Theorem 1.

Proof. Let $t = e$. Then the binary representation of t is:

$$t = \overbrace{00\cdots01}^{s}\overbrace{00\cdots01}^{s}\overbrace{00\cdots01}^{s}\overbrace{00\cdots00}^{s}\overbrace{11\cdots11}^{s}$$

Using the proof techniques from Lemmas 3, 4 and 5 it can easily be verified that maximum value of $H(r + t2^{-i})$ is $l + \lceil \frac{n}{l} \rceil + 2$, where $0 \leq i \leq m - 1$ and r is defined in Theorem 1. Now the proof follows directly from Lemma 1, and Proposition 1. □

Note that the degree of $f(x)$ is very small. From Eqn. (5.2) this bound is only useful if $n \geq 100$. This might not be of practical significance at present but we have provided the bound for the sake of theoretical interest.

5.9 Comparative Study of our $\mathcal{AI}$ Bound

Now we compare our bound on the $\mathcal{AI}$ with the bound given in Fact 1, and the exact $\mathcal{AI}$, obtained experimentally, in [2]. Table 5.1 shows how the upper bound on the $\mathcal{AI}$ of monomial trace functions with Inverse, Kasami and Niho exponents decreases as n increases. Table 5.2 compares our bound with the exact $\mathcal{AI}$ obtained in [2]. For $n = 16$ and 20 (a * in table) functions with Kasami-like exponents were modified in [2] to make them balanced. Therefore for $n = 16$ and 20 comparison with our bound is not valid. Note that for $n \leq 20$ our bound is tight for inverse exponent and is very close to the exact $\mathcal{AI}$ for Kasami exponent. Also the experimental results suggest that $\mathcal{AI}$ of Kasami exponent is better than inverse exponent. However it is clear from Theorems 2 and 3 that the difference between the upper bound on their $\mathcal{AI}$ is at most 2 irrespective of the value of n.

Table 5.1: $\mathcal{AI}$ bounds for Inverse, Kasami and Niho functions

f	n	$deg(f)$	Bound from Fact 1 ($\lceil \frac{n}{2} \rceil$)	Our bound on $\mathcal{AI}$
Inverse	16	15	8	6
	36	35	18	10
	100	99	50	18
Kasami	16	8	8	8
	36	18	18	12
	100	50	50	20
Niho	15	8	8	8
	35	18	18	12
	99	50	50	20

Table 5.2: Comparison of our $\mathcal{AI}$ bound with experimental results from [2]

n	Inverse		Kasami		
	Exact $\mathcal{AI}$	Our bound	s	Exact $\mathcal{AI}$	Our bound
13	6	6	5	6	8
14	6	6	6	6	8
15	6	6	6	7	8
16	6	6	7	7^*	8
17	7	7	7	8	9
18	7	7	8	8	9
19	7	7	8	9	9
20	7	7	9	9^*	8
100	X	18	49	X	20

5.10 Generalization to Polynomial Functions

Any balanced polynomial function $f(x)$ can be represented by Eqn. (2.2) which is reproduced here as

$$f(x) = \sum_{k \in \Gamma(n)} Tr_1^{n_k}(A_k x^k), A_k \in \mathbb{F}_{2^{n_k}}, x \in \mathbb{F}_{2^n},$$

which is simply a sum of monomial trace functions. If we only consider monomial trace functions as multipliers the result in Theorem 1 can be generalized to all balanced polynomial functions. Let u_k be the number of runs of 1's in the binary representation of k and $u = \max_{k\in\Gamma(n)}\{u_k\}$ such that $A_k \neq 0$. Let $g(x)$ be a monomial trace function defined in Theorem 1. Then

$$deg(f(x)g(x)) \leq ul + \left\lceil \frac{n}{l} \right\rceil - 1.$$

To obtain a meaningful bound on $\mathcal{AI}$ of f, u must satisfy Eqn. (5.2). The above result implies that $\mathcal{AI}$ of f is upper bounded by the maximum $\mathcal{AI}$ of the single trace functions in the polynomial representation of f.

5.11 $\mathcal{AI}$ of WG and Hyper-bent Functions

So far in this chapter we have restricted our discussion to the $\mathcal{AI}$ of monomial trace functions. In Section 5.10 we generalized our results to all polynomial functions however this bound is not tight. In the absence of tight bounds for general polynomial functions we present some experimental results on the $\mathcal{AI}$ of two other cryptographically significant functions, i.e., WG and Hyper-bent.

We begin with the definition of a bent function. A bent function, f, is a function from $\mathbb{F}_{2^n} \rightarrow \mathbb{F}_2$ for which [52]

$$\hat{f}(\lambda) = \sum_{x\in\mathbb{F}_{2^n}} (-1)^{Tr(\lambda x)+f(x)} = \pm\sqrt{2^n}, \forall\lambda \in \mathbb{F}_{2^n}$$

where n is even and we write $n = 2m$. A bent function is called a hyper-bent function [116] if,

$$\hat{f}(\lambda, c) = \pm 2^m, \forall\lambda \in \mathbb{F}_{2^n}, c : 0 < c < 2^n - 1, gcd(c, 2^n - 1) = 1$$

where

$$\hat{f}(\lambda, c) = \sum_{x\in\mathbb{F}_{2^n}} (-1)^{Tr(\lambda x^c)+f(x)}.$$

Table 5.3: $\mathcal{AI}$ of WG tranformations

n	$deg(f)$	Total WG transformations	No. of transformations whose degree can be reduced	$deg(g)$	$deg(fg)$
7	4	18	0	-	-
8	4	16	1	3	3
10	5	60	2	4	4
11	5	176	0	-	-
13	6	631	0	-	-

Hyper-bent functions are important cryptographic functions as they have the maximum nonlinearity (minimum distance from all affine functions). A hyper-bent function f in n variables has degree $d = n/2$ where n is even. We have used the algorithm given in [55] to test hyper-bent functions for algebraic immunity. Our exhaustive search shows that all hyper-bent functions in 6 variables($n = 6$ and $d = 3$) have maximum $\mathcal{AI}$. However our results for $n = 8$ and $n = 10$ show that for about 10 percent of the hyper-bent functions $\mathcal{AI} = n - 1$. We also tested some functions with $n = 12$ but were not able to find a Hyper-bent function with $\mathcal{AI}$ less than $n - 1$.

We introduced WG transformations and their cryptographic properties in Chapter 3. These transformations are balanced with high linear complexity and are first order resilient. The degree of a WG function in n variables is $\lceil n/3 \rceil + 1$. We tested all WG functions in n variables where $7 \leq n \leq 13$. The results are shown in Table 5.3. For each n we list the degree of the transformation, total number of WG transformations, the degree of low degree multiplier g, if it exists, and the degree of product $f * g$. It is clear from the table that for $n = 8$ there is only one transformation whose degree can be reduced by a multiplier g of degree 3. Similarly for $n = 10$, two transformations exist whose degree can be reduced by multipliers of degree 4. For n=7, 11, and 13 all WG transformations have $\mathcal{AI}$ equal to their algebraic degree.

5.12 Conclusions

In this chapter we used the theory of polynomial functions to provide an upper bound on $\mathcal{AI}$ of monomial trace functions. The low degree multiples were also obtained directly from the formula for any n. This is particularly useful because so far there are no known efficient algorithms to find the $\mathcal{AI}$ of a function with large number of inputs. We improved the $\mathcal{AI}$ bound on inverse, Kasami, Niho and Dobbertin functions and showed that their $\mathcal{AI}$ is very low. We also generalized our results to polynomial functions. Finally we presented experimental results on the $\mathcal{AI}$ of WG transformations and Hyper-bent functions. An interesting extension of this research is to determine the lower bound on the $\mathcal{AI}$ of monomial trace functions. However it seems to be more challenging as it requires proving the non-existence of low degree multipliers.

Chapter 6

Algebraic Immunity of S-boxes

In this chapter we investigate the resistance of S-boxes against algebraic attacks. Courtois defined this resistance in terms of the algebraic immunity of an S-box which depends on the number and type of linearly independent multivariate equations it satisfies. This definition is controversial and some researchers believe that the existence of such equations does not imply that a block cipher is vulnerable to algebraic attacks. They argue that the system of multivariate equations obtained in these algebraic attacks is hard to solve. On the other hand some algorithms [25,39,40] have been successful in solving such systems but on a smaller scale. These algorithms are mostly based on heuristics and it is difficult to say how well they will scale to larger systems such as the one for AES. We believe that although Courtois definition may not measure the complexity of solving such systems accurately, the existence of these multivariate equations indicate an algebraic weakness in the design of the S-box. Simple and abundant algebraic relations between the input and output of nonlinear transformations are not desirable. Therefore we will use the term algebraic immunity to refer to the absence of low degree multivariate equations between the input and output of an S-box.

In this chapter our main focus is to develop techniques to find the number of linearly independent, multivariate, bi-affine and quadratic equations for S-boxes based on power mappings. These techniques can be used to obtain the exact number of equations for any class of power mappings. We present two algorithms to calcu-

late the number of bi-affine and quadratic equations for any (n, n) S-box based on power mapping. The time complexity of both algorithms is only $O(n^2)$. To design algebraically immune S-boxes we present four new classes of S-boxes that guarantee zero bi-affine equations and one class of S-boxes that guarantees zero quadratic equations. We also discuss the algebraic immunity of power mappings based on Kasami, Niho, Dobbertin, Gold, Welch and Inverse exponents.

To design cryptographically strong S-boxes we also consider their resistance against other powerful attacks, i.e., linear and differential cryptanalysis. We provide two new conjectures about the nonlinearity and differentially k-uniform property of Kasami and Kasami like exponents. Finally we solve an open problem to find an (n, n) bijective nonlinear S-box with more than $5n$ quadratic equations and conjecture that the upper bound on this number is $\frac{n(n-1)}{2}$.

6.1 Background

We begin by reviewing the developments in the field of algebraic attacks on block ciphers. The first algebraic attack on a block cipher was discussed in [108]. For recent developments in the area of algebraic attacks on block ciphers see [5, 11, 21, 25, 80, 81]. In [25] Courtois and Pieprzyk showed that AES [26] can be attacked by solving an overdefined system of algebraic equations. This is possible because the only nonlinear component in AES, i.e., the S-box, can be expressed as an overdefined system of algebraic equations. The authors presented an algorithm called XSL to solve this system of multivariate equations and also introduced the notion of algebraic immunity, Υ, of S-boxes where Υ is an important parameter in measuring the complexity of the XSL algorithm. For a $\mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n}$ S-box Υ is defined as $\Upsilon = ((t-r)/n)^{\lceil (t-r)/n \rceil}$, where r is the number of equations and t is the number of monomials in these equations. A lower value of r means a higher value of Υ and therefore higher complexity of the algebraic attack. In [25] the authors showed that for AES S-box $\Upsilon = 2^{22.9}$ and claimed that for secure ciphers Υ should be greater than 2^{32}.

Inspired by [25] Cheon and Lee [18] developed tools to calculate the number

of linearly independent multivariate equations for algebraic S-boxes. They used their results to estimate the algebraic immunity Υ of maximally nonlinear power S-boxes (based on Gold, Kasami and inverse exponents [34]). However Courtois et al. disputed their results in [22] by showing that in most cases the number of linearly independent multivariate equations calculated by them are incorrect. This was done by experimentally finding the total number of quadratic equations for Gold and Kasami power S-boxes. Power mappings are of interest because unlike random permutations, they can be implemented in hardware without a lookup table. This facilitates compact and fast implementations of S-boxes in hardware.

In [22, Appendix A] Courtois et al. also used the polynomial representation of the algebraic S-boxes to prove that S-boxes based on inverse mapping in $\mathbb{F}_{2^n}$ have $3n-1$ bi-affine equations and $5n-1$ quadratic equations. However they did not generalize their results to other power S-boxes and only provided experimental results for S-boxes based on Gold, Dobbertin, Niho, Welch and Kasami exponents. The largest S-box experimentally tested by them was $n = 17$. In this chapter we use the polynomial representations to develop techniques which can be used to obtain the exact number of bi-affine and quadratic equations for any class of power mappings. From these techniques we also develop two very simple algorithms which, given an (n, n) S-box and the exponent of power mapping, calculate the exact number of bi-affine and quadratic equations respectively. The time complexity of both algorithms is $O(n^2)$ which is very small even for very large S-boxes (for example $n = 2^{20}$). The algorithms currently available in the literature to find such equations have time complexities that are exponential in n and therefore impractical for $n > 25$. Note however that these algorithms find the actual equations whereas our algorithms only calculate the number of such equations. Towards designing S-boxes with highest algebraic immunity, we provide four classes of power S-boxes that guarantee zero bi-affine equations and one class that guarantees zero quadratic equations. We examine other cryptographic properties, i.e., nonlinearity, algebraic degree and uniform differential property of these S-boxes and list cryptographically strong power S-boxes for $n = 8$ and 10. We also provide two new conjectures regarding the nonlinearity and uniform differential property of Kasami like power

mappings. In addition to this we solve an open problem given in [25, 22] to find a bijective nonlinear (n, n) S-box with more than $5n$ quadratic equations. We conjecture the upper bound on this number to be $\frac{n(n-1)}{2}$ for even n.

6.2 Bi-affine and Quadratic Equations for Power Mappings

Let's fix any arbitrary basis $\{\alpha_0, \alpha_1, \cdots, \alpha_{n-1}\}$ for $\mathbb{F}_{2^n}$. Then $\mathbb{F}_{2^n}$ and $\mathbb{F}_2^n$ are isomorphic and can be used interchangeably. Consider $F : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n}$ to be an S-box based on a power mapping. Such S-boxes are classified according to the exponent a of the power mapping such that $y = F(x) = x^a$.

Proposition 2 *Let $h(x, y) = \sum_{i,j} a_{i,j} x_i y_j$ be a bi-affine Boolean function in $2n$ variables, where $x = \sum_{i=0}^{n-1} x_i \alpha_i$, $y = \sum_{i=0}^{n-1} y_i \alpha_i$ and $a_{i,j} \in \mathbb{F}_2$. Then $h(x, y)$ has a unique polynomial representation in $\mathbb{F}_{2^n}$ such that*

$$h(x, y) = \sum_{k=0}^{n-1} Tr_1^n(b_k x^{2^k} y),\ b_k \in \mathbb{F}_{2^n}.$$

Proof. Let $\{\alpha_0, ..., \alpha_{n-1}\}$ and $\{\beta_0, ..., \beta_{n-1}\}$ be dual bases of $\mathbb{F}_{2^n}$ and $x = x_0\alpha_0 + \cdots + x_{n-1}\alpha_{n-1}$ and $y = y_0\alpha_0 + \cdots + y_{n-1}\alpha_{n-1}$. Now we can write $x_i = Tr_1^n(\beta_i x)$ and $y_j = Tr_1^n(\beta_j y)$.

Therefore

$$\begin{aligned} h(x,y) &= \sum_{i,j} a_{i,j}x_iy_j = \sum_{i,j} a_{i,j}Tr_1^n(\beta_i x)Tr_1^n(\beta_j y) \\ &= \sum_{i,j} a_{i,j}\sum_{k=0}^{n-1} Tr_1^n(\beta_i^{2^k}\beta_j x^{2^k}y) \\ &= \sum_{k=0}^{n-1} Tr_1^n(\sum_{i,j} a_{i,j}\beta_i^{2^k}\beta_j x^{2^k}y) \\ &= \sum_{k=0}^{n-1} Tr_1^n(b_k x^{2^k}y) \text{ where } b_k = \sum_{i,j} a_{i,j}\beta_i^{2^k}\beta_j \end{aligned}$$

□

Corollary 1 *Let $y = x^a$ and $h(x,y) = \sum_{i,j} a_{i,j}x_iy_j + \sum_j v_jy_j$ be a Boolean function in $2n$ variables where $a_{i,j}, v_j \in \mathbb{F}_2$. Then $h(x,y)$ has a unique polynomial representation in $\mathbb{F}_{2^n}$ such that*

$$h(x,y) = \sum_{k=0}^{n-1} Tr_1^n(b_k x^{2^k+a}) + Tr_1^n(cx^a),\ b_k, c \in \mathbb{F}_{2^n}.$$

Proof. From proposition 2 we have $\sum_{i,j} a_{i,j}x_iy_j = \sum_{k=0}^{n-1} Tr_1^n(b_k x^{2^k}y)$ and we know $\sum_j v_jy_j = Tr_1^n(\sum_j v_j\beta_j y) = Tr_1^n(cy)$. Putting $y = x^a$ we have the proof.

□

Proposition 3 *Let $h(y) = \sum_{i\le j} a_{i,j}y_iy_j$ be a Boolean function in n variables, where $a_{i,j} \in \mathbb{F}_2$. Then $h(y)$ has a unique polynomial representation in $\mathbb{F}_{2^n}$ such that:*

- *if n is even, $n = 2m$:*

$$h(y) = Tr_1^n(d_0y) + \sum_{k=1}^{m-1} Tr_1^n(d_k y^{(2^k+1)}) + Tr_1^m(d_m y^{(2^m+1)}),\ d_0, d_k \in \mathbb{F}_{2^n}, d_m \in \mathbb{F}_{2^m}.$$

- *if n is odd, $n = 2m + 1$:*

$$h(y) = Tr_1^n(d_0 y) + \sum_{k=1}^{m} Tr_1^n(d_k y^{(2^k+1)}),\ d_0, d_k \in \mathbb{F}_{2^n}.$$

Proof. Let $\{\alpha_0, ..., \alpha_{n-1}\}$ and $\{\beta_0, ..., \beta_{n-1}\}$ be dual bases of $\mathbb{F}_{2^n}$, $x = x_0\alpha_0 + \cdots + x_{n-1}\alpha_{n-1}$, and $y = y_0\alpha_0 + \cdots + y_{n-1}\alpha_{n-1}$. Now we can write $y_j = Tr_1^n(\beta_j y)$.

Therefore

$$\begin{aligned} h(y) &= \sum_{i\le j} a_{i,j} y_i y_j = \sum_{i\le j} a_{i,j} Tr_1^n(\beta_i y) Tr_1^n(\beta_j y) \\ &= \sum_{i\le j} a_{i,j} \sum_{k=0}^{n-1} Tr_1^n(\beta_i^{2^k} \beta_j y^{(2^k+1)}) \\ &= \sum_{k=0}^{n-1} Tr_1^n(\sum_{i\le j} a_{i,j} \beta_i^{2^k} \beta_j y^{(2^k+1)}) \end{aligned}$$

Now consider the case when n is even. Note that y and y^2 belong to the same coset. Also for $1 \le k < m$, $y^{(2^k+1)}$ and $y^{(2^{n-k}+1)}$ belong to the same coset. Note that $2^m + 1$ belongs to the coset of length m, i.e., $|C_{2^m+1}| = m$. Therefore

$$h(y) = Tr_1^n(d_0 y) + \sum_{k=1}^{m-1} Tr_1^n(d_k y^{(2^k+1)}) + Tr_1^m(d_m y^{(2^m+1)}),$$

where $d_0 = \sum_{i\le j} a_{i,j}(\beta_i\beta_j)^{2^{n-1}}$, $d_k = \sum_{i\le j} a_{i,j}(\beta_i^{2^k}\beta_j + \beta_i\beta_j^{2^k})$, $1 \le k < m$, and $d_m = \sum_{i\le j} a_{i,j}\beta_i^{2^m}\beta_j$. The proof for n odd is similar. □

Corollary 2 *Let $y = x^a$ and $h(x, y) = \sum_{i,j} a_{i,j} x_i y_j + \sum_{i\le j} e_{i,j} y_i y_j + \sum_i v_i y_i$ be a quadratic Boolean function in n variables where $a_{i,j}, e_{i,j}, v_j \in \mathbb{F}_2$. Then $h(x, y)$ has a unique polynomial representation in $\mathbb{F}_{2^n}$ such that:*

- *if n is even, $n = 2m$:*

$$h(x,y) = \sum_{k=0}^{n-1} Tr_1^n(b_k x^{2^k+a}) + Tr_1^n(c'x^a) + \sum_{k=1}^{m-1} Tr_1^n(d_k x^{(2^k+1)a}) + Tr_1^m(d_m x^{(2^m+1)a}),$$

$b_k, d_k, c' \in \mathbb{F}_{2^n}, d_m \in \mathbb{F}_{2^m}$.

- *if n is odd, $n = 2m+1$:*

$$h(x,y) = \sum_{k=0}^{n-1} Tr_1^n(b_k x^{2^k+a}) + Tr_1^n(c'x^a) + \sum_{k=1}^{m} Tr_1^n(d_k x^{(2^k+1)a}), b_k, d_k, c' \in \mathbb{F}_{2^n}.$$

Proof. From Corollary 1 we have

$$\sum_{i,j} a_{i,j} x_i y_j + \sum_j v_j y_j = \sum_{k=0}^{n-1} Tr_1^n(b_k x^{2^k+a}) + Tr_1^n(cx^a).$$

From proposition 3, for even n, we have

$$\sum_{i \le j} a_{i,j} y_i y_j = Tr_1^n(d_0 x^a) + \sum_{k=1}^{m-1} Tr_1^n(d_k y^{(2^k+1)}) + Tr_1^m(d_m y^{(2^m+1)}).$$

Now we have

$$\sum_{i,j} a_{i,j} x_i y_j + \sum_j v_j y_j + \sum_{i \le j} a_{i,j} y_i y_j = \sum_{k=0}^{n-1} Tr_1^n(b_k x^{2^k+a}) + Tr_1^n((c+d_0)x^a)$$

$$+ \sum_{k=1}^{m-1} Tr_1^n(d_k x^{(2^k+1)a}) + Tr_1^m(d_m x^{(2^m+1)a}),$$

where $c' = c + d_0$. The proof for n odd is similar. □

6.2.1 Bi-affine Equations

For a given power mapping $y = x^a$ we want to find the number of bi-affine equations of the form

$$\sum_{i,j} a_{i,j} x_i y_j + \sum_j v_j y_j + \sum_i u_i x_i + z = 0, \quad a_{i,j}, v_j, u_i, z \in \mathbb{F}_2. \tag{6.1}$$

Let $h(x, y)$ be a bilinear function as defined in Corollary 1. Then

$$\begin{aligned} h(x,y) &= \sum_{k=0}^{n-1} Tr_1^n(b_k x^{2^k+a}) + Tr_1^n(cx^a), \quad b_k, c \in \mathbb{F}_{2^m} \\ &= \sum_i u_i x_i + z. \end{aligned} \tag{6.2}$$

Therefore the number of bi-affine equations, of the form as given in (6.1), is equal to the number of functions $h(x, y)$ that are affine (i.e.,$\sum_i u_i x_i + z$).

$h(x, y)$ in (6.2) will be affine in the following cases:

1. If for any $0 \leq k \leq n-1$, $H(2^k + a) = 1$ then $Tr_1^n(b_k x^{2^k+a})$ will give 2^n affine functions (as $b_k \in \mathbb{F}_{2^n}$). Note that only n of these functions are linearly independent. Similarly if $H(a) = 1$ we will get n linearly independent functions.
2. If for any $0 \leq k \leq n-1$, $H(2^k + a) > 1$ and $2^k + a \in C_{k'}$ where $|C_{k'}| = m' < n$, $m'|n$, then $Tr_1^n(b_k x^{2^k+a})$ will give $2^{n-m'}$ affine functions. Note that $Tr_1^n(b_k x^{2^k+a}) = Tr_1^{m'}(Tr_{m'}^n(b_k x^{2^k+a})) = Tr_1^{m'}(x^{2^k+a} Tr_{m'}^n(b_k))$. Now $Tr_{m'}^n(b_k)$ must be 0 for $Tr_1^n(b_k x^{2^k+a})$ to be an affine function. Therefore we get $2^{n-m'}$ affine functions. Only $n - m'$ of these functions are linearly independent. The same argument holds for $Tr_1^n(cx^a)$.
3. Let $\mathcal{A} = \{2^k + a : 0 \leq k \leq n-1, H(2^k + a) > 1 \wedge 2^k + a \notin \mathbb{F}_{2^{m''}}, m'' < n\}$. Now we define $\mathcal{S}$, the set of exponents such that

$$\mathcal{S} = \begin{cases} \mathcal{A} \cup \{a\}, & \text{if } H(a) > 1 \wedge a \notin \mathbb{F}_{2^{m'}}, m' < n \\ \mathcal{A}, & \text{otherwise.} \end{cases}$$

Note that $\mathcal{S}$ can be a multiset. If any two exponents from $\mathcal{S}$ belong to the same

coset then they will give $2^{(2-1)n} = 2^n$ affine functions. For example if $2^{k_1} + a$ and $2^{k_2} + a$ $(k_1 \neq k_2)$ belong to the same coset then we can write $2^{k_1} + a = (2^{k_2} + a)2^l$ where $1 \leq l \leq n$. Now we have a term $Tr_1^n((b_{k_1} + (b_{k_2})^{2^l})x^{2^{k_1}+a})$ where $b_{k_1} + (b_{k_2})^{2^l}$ can be zero in 2^n ways. In general if there exist t exponents that belong to the same coset , we will have $2^{(t-1)n}$ affine functions.

4. If $a \in C_{2^{n-1}-1}$ then $2^k + a = 2^n - 1$ for some $0 \leq k \leq n-1$. Then we have $Tr_1^n(b_k x^{2^n-1}) = Tr_1^n(b_k)$ which can be 0 in 2^{n-1} ways. Therefore we have $n-1$ linearly independent affine functions. Note $C_{2^{n-1}-1}$ is the only coset with Hamming weight $n-1$ and the inverse exponent belongs to this coset.

Based on the above 4 cases, we now give Algorithm 1 to compute the total number of linearly independent bi-affine equations for a given power mapping $y = x^a$ over $\mathbb{F}_{2^n}$.

Algorithm 1 Computing number of linearly independent bi-affine equations

Input a, n.

Output Total number of independent bi-affine equations.

1: For given a and $2^k + a, 0 \leq k \leq n-1$, compute their coset leaders in array cst_l and their coset sizes in array cst_s;
2: sort array cst_l in ascending order and shuffle array cst_s accordingly;
3: $k \leftarrow 0, eqnum \leftarrow 0$;
4: **while** $k \leq n$ **do**
5: if$(H(cst_l[k] = 0))$: $eqnum \leftarrow eqnum + (n-1)$;
6: elseif$(H(cst_l[k] = 1))$: $eqnum \leftarrow eqnum + n$;
7: elseif$(cst_s[k] < n))$: $eqnum \leftarrow eqnum + (n - cst\_s[k])$;
8: else
9: **while** $cst\_l[k] = cst\_l[k+1]$ **do**
10: $eqnum \leftarrow eqnum + n$; $k \leftarrow k + 1$;
11: $k \leftarrow k + 1$;

Theorem 6 *Let $y = x^a$ be a power mapping over $\mathbb{F}_{2^n}$. Then using Algorithm 1, the total number of linearly independent bi-affine equations can be computed in time $O(n^2)$.*

Proof. In Algorithm 1, step 1 takes time $O(n^2)$. Step 2 takes time $O(nlog(n))$ and

steps 4-10 take time $O(n)$. Therefore total time complexity is $O(n^2)$. Correctness of Algorithm 1 follows from the 4 cases discussed above.

6.2.2 Quadratic Equations

For a given power mapping $y = x^a$ we want to find the number of quadratic equations of the form

$$\sum_{i,j} a_{i,j}x_iy_j+\sum_{i\leq j} b_{i,j}y_iy_j+\sum_i v_iy_i+\sum_{i\leq j} c_{i,j}x_ix_j+\sum_i u_ix_i+z=0, \quad a_{i,j},b_{i,j},c_{i,j},v_i,u_i,z\in\mathbb{F}_2. \tag{6.3}$$

Let $h(x,y)$ be a quadratic function as defined in Corollary 2. Then

- if n is even, $n=2m$:

$$\begin{aligned} h(x,y) &= \sum_{k=0}^{n-1} Tr_1^n(b_kx^{2^k+a})+Tr_1^n(c'x^a)+\sum_{k=1}^{m-1} Tr_1^n(d_kx^{(2^k+1)a}) \\ &\quad +Tr_1^m(d_mx^{(2^m+1)a}) \\ &= \sum_{i\leq j} c_{i,j}x_ix_j+\sum_i u_ix_i+z. \end{aligned} \tag{6.4}$$

- if n is odd, $n=2m+1$:

$$\begin{aligned} h(x,y) &= \sum_{k=0}^{n-1} Tr_1^n(b_kx^{2^k+a})+Tr_1^n(c'x^a)+\sum_{k=1}^{m} Tr_1^n(d_kx^{(2^k+1)a}) \\ &= \sum_{i\leq j} c_{i,j}x_ix_j+\sum_i u_ix_i+z. \end{aligned}$$

Therefore the number of quadratic equations, of the form as given in (6.3), is equal to the number of functions $h(x,y)$ that are quadratic (i.e.,$\sum_{i\leq j} c_{i,j}x_ix_j+\sum_i u_ix_i+z$). Consider $n=2m$ to be even then $h(x,y)$ in (6.4) will be quadratic in the following cases:

1. If for any $0\leq k\leq n-1, 1\leq H(2^k+a)\leq 2$ then $Tr_1^n(b_kx^{2^k+a})$ will give n linearly independent quadratic functions. If for any $0\leq k\leq n-1, 1\leq H((2^k+1)a)\leq$

2 then $Tr_1^n(d_k x^{(2^k+1)a})$ will give n linearly independent quadratic functions. If $1 \leq H(a) \leq 2$ then $Tr_1^n(c'x^a)$ will give n linearly independent quadratic functions. If $\leq H((2^m+1)a) \leq 2$ then $Tr_1^m(d_m x^{(2^m+1)a})$ will give m linearly independent quadratic functions (as $d_m \in \mathbb{F}_{2^m}$).

2. If for any $0 \leq k \leq n-1$, $H(2^k+a) > 2$ and $2^k+a \in C_{k_1}$ where $|C_{k_1}| = m_1 < n$, $m_1|n$, then $Tr_1^n(b_k x^{2^k+a})$ will give 2^{n-m_1} quadratic functions. If for any $1 \leq k \leq m-1$, $H((2^k+1)a) > 2$ and $(2^k+1)a \in C_{k_2}$ where $|C_{k_2}| = m_2 < n$, $m_2|n$, then $Tr_1^n(d_k x^{(2^k+1)a})$ will give 2^{n-m_2} quadratic functions. If $H(a) > 2$ and $a \in C_{k_3}$ where $|C_{k_3}| = m_3 < n$, $m_3|n$, then $Tr_1^n(c'x^a)$ will give 2^{n-m_3} quadratic functions. If $H((2^m+1)a) > 2$ and $(2^m+1)a \in C_{k_4}$ where $|C_{k_4}| = m_4 < m$, then $Tr_1^m(d_m x^{(2^m+1)a})$ will give 2^{m-m_4} quadratic functions.
3. Let

$$\mathcal{B} = \{2^k + a : 0 \leq k \leq n-1, H(2^k+a) > 2 \wedge 2^k + a \notin \mathbb{F}_{2^{m_5}}, m_5 < n\}$$

$$\cup\{(2^k+1)a : 1 \leq k \leq m-1, H((2^k+1)a) > 2 \wedge (2^k+1)a \notin \mathbb{F}_{2^{m_6}}, m_6 < n\}.$$

   Now we define the set of exponents $\mathcal{T}$ such that

$$\mathcal{T} = \begin{cases} \mathcal{B} \cup \{a\}, & \text{if } H(a) > 2 \wedge a \notin \mathbb{F}_{2^{m_7}}, m_7 < n \\ \mathcal{B}, & \text{otherwise} \end{cases}$$

   Note $\mathcal{T}$ can be a multiset. If any two exponents from $\mathcal{T}$ belong to the same coset then they will give $2^{(2-1)n} = 2^n$ quadratic functions. In general if there exist t exponents that belong to the same coset, we will have $2^{(t-1)n}$ quadratic functions.
4. If $a \in C_{2^{n-1}-1}$ then $2^k + a = 2^n - 1$ for some $0 \leq k \leq n-1$. Therefore we have $n-1$ linearly independent quadratic functions.

Note that odd n can be handled in the similar manner and Algorithm 2 can be modified accordingly. Based on the above 4 cases, we now give Algorithm 2 to compute the total number of linearly independent quadratic equations for a given power mapping $y = x^a$ over $\mathbb{F}_{2^n}$ when n is even.

Algorithm 2 Computing number of linearly independent quadratic equations

Input $a, n = 2m$.

Output Total number of independent bi-affine equations.

1: For given a and $2^k + a, 0 \leq k \leq n-1$, and $(2k+1)a, 1 \leq k \leq m-1$, compute their coset leaders in array cst_l and their coset sizes in array cst_s;
2: Compute coset leader of $(2^m+1)a$ in $cstm\_l$ and its coset size in $cstm\_s$
3: sort array cst_l in ascending order and shuffle array cst_s accordingly;
4: $k \leftarrow 0, eqnum \leftarrow 0$;
5: **while** $k \leq n+m-1$ **do**
6: if$(H(cst_l[k] = 0))$: $eqnum \leftarrow eqnum + (n-1)$;
7: elseif$(H(cst_l[k] \leq 2))$: $eqnum \leftarrow eqnum + n$;
8: elseif$(cst_s[k] < n))$: $eqnum \leftarrow eqnum + (n - cst\_s[k])$;
9: else
10: **while** $cst\_l[k] = cst\_l[k+1]$ **do**
11: $eqnum \leftarrow eqnum + n$**;** $k \leftarrow k+1$**;**
12: $k \leftarrow k+1$**;**
13: if$(H(cstm_l) \leq 2)$:$eqnum \leftarrow eqnum + n$;
14: elseif$(cstm_s < m)$:$eqnum \leftarrow eqnum + (m - cstm\_s)$;

From the above discussion we have the following theorem.

Theorem 7 *Let $y = x^a$ be a power mapping over $\mathbb{F}_{2^n}$. Then using Algorithm 2, the total number of linearly independent quadratic equations can be computed in time $O(n^2)$.*

6.3 $\mathcal{AI}$ of Cryptographically Significant Power Mappings

S-boxes which satisfy zero bi-affine and/or quadratic equations provide optimal resistance against algebraic attacks and therefore are of great interest. In this section we provide several S-box constructions that satisfy this criteria. A cryptographically strong S-box must also have high nonlinearity, good uniform differential property, and high algebraic degree to resist linear, differential, and higher order differential attacks respectively. Unfortunately there are very few S-boxes that satisfy all the

above requirements and as n increases, finding these S-boxes becomes extremely difficult. Therefore we identify classes of S-boxes which provably have all the above mentioned properties. We also provide experimental results for smaller values of n to identify good S-boxes and show various tradeoffs involved in the selection of an S-box.

6.3.1 Power Mappings with Zero Bi-affine Equations

It is easy to see that the probability of a randomly chosen S-box having zero bi-affine equations is high if n is large [25]. This holds true for S-boxes based on power mappings as well. For example for $n = 8$, 18.8 percent power mappings have zero bi-affine equations. For $n = 16$, 91.1 percent and for $n = 25$, 99.9 percent power mappings satisfy this condition. However note that several highly nonlinear S-boxes (for example inverse mapping) always have bi-affine equations and therefore it is important to calculate the exact number of bi-affine equations an S-box satisfies. If a power mapping is selected randomly then we can use Algorithm 1 in Section 6.2.1 to find the number of bi-affine equations it satisfies. Otherwise any one of the following S-box constructions can be used. Consider the power mapping $y = x^a$ over $\mathbb{F}_{2^n}$. From Section 6.2.1, the three necessary and sufficient conditions for the power mapping to have zero bi-affine equations are:

1. $H(a) > 1$ and $H(2^i + a) > 1, 0 \leq i \leq n-1$.
2. $(2^i + a)2^l \not\equiv (2^i + a) \bmod 2^n - 1, 0 \leq i \leq n-1$ and $a2^l \not\equiv a \bmod 2^n - 1, l < n$.
3. $(2^i + a)2^l \not\equiv (2^j + a) \bmod 2^n - 1, i \neq j, 0 \leq i, j \leq n-1$ and $a2^l \not\equiv (2^k + a) \bmod 2^n - 1, 0 \leq k \leq n-1, l < n$.

Now we provide four (n, n) power S-box constructions which have zero bi-affine equations. The first construction, given in Theorem 8, consists of S-box based on Kasami-like exponents (see Section 6.3.3). A subclass of these exponenets, called Kasami exponents was studied by Dobbertin in [34] and mappings based on these exponents were shown to be APN.

Theorem 8 *Let $n = 2m, n \geq 8$ and $a = 2^{m+1} + 2^{m-1} - 1$. Then power mapping $y = x^a$ over $\mathbb{F}_{2^n}$ has no bi-affine equations.*

Proof. Consider the binary representation of a and 2^i.

$$\begin{aligned} a &= \overbrace{00\cdots010}^{m}\overbrace{011\cdots11}^{m} \\ 2^i &= \underbrace{00\cdots\cdots001}\underbrace{0\cdots0}_{i} \end{aligned}$$

1. $H(a) > 1$ and from the binary representation of a it is obvious that $H(2^i + a) > 1, 0 \leq i \leq n-1$.
2. If $2^i + a, \in C_z, 0 \leq i \leq n-1$ where $|C_z| < n$ then $|C_z|$ divides n. This means that binary representation of $2^i + a$ must contain at least 2 runs of 1's of the same length and 2 runs of 0's of same length. We have 2 runs of 1's when $i = 0, m-1, m, m+1, m+2, 2m-1$. However in all these cases the lengths of the 2 run's of 1's is always different. For all the other cases we get 3 runs of 1's, however they will never be of the same length (we always have 1 run of length 1 and 1 run of length more than 1). Therefore $|C_z| = n$ and $(2^i + a)2^l \not\equiv (2^i + a), l < n$. Also it is evident from the binary representation of a that $a2^l \not\equiv (2^k + a), k < l$.
3. We know that if two integers b and c belong to the same coset then some cyclic shift of binary representation of b gives c. $H(2^i + a) = i + 2, 0 \leq i \leq m-1$, i.e., all belong to distinct cosets. $H(2^i + a) = m, i = m-2, m+1$ but the number of runs of 1's is different in the two cases. $H(2^i + a) = m+1, m \leq i \leq 2m-1, i \neq m+1$. Although the Hamming weight in all these cases is same but either the run's of 1's and 0's are of different length or they appear in different order. Therefore it is not possible to get one integer from the cyclic shift of the binary representation of another element. Therefore for each $0 \leq i \leq n-1$, $2^i + a$ belongs to a distinct coset. Also $H(a) = H(2^i + a)$ only when $i = m-2, m+1$, however in both cases the length of run's of 0's is different from a.

The three necessary and sufficient conditions for $y = x^a$ to have zero bi-affine equations (from Section 6.3.1) are satisfied. □

Note that this mapping is maximally nonlinear, has high algebraic degree and good uniform differential property (See Section 6.3.3).

Theorem 9 *Let $n \geq 8$ and $a = 1 + 2 + \sum_{i=3}^{r} 2^i, 3 \leq r \leq n-3$. Then power mapping $y = x^a$ over $\mathbb{F}_{2^n}$ has zero bi-affine equations except when $n = 8, r = 5$.*

Proof. Consider the binary representation of a

$$a = \overbrace{00}^{2}\overbrace{0\cdots0}\overbrace{1\cdots1011}^{r+1}$$

1. $H(a) > 1$ and from the binary representation of a, $H(2^i+a) > 1, 0 \leq i \leq n-1$.
2. If $2^i + a, \in C_z, 0 \leq i \leq n-1$ where $|C_z| < n$ then $|C_z|$ divides n. This means that for even n binary representation of 2^i+a must contain at least 2 runs of 1's of the same length and 2 runs of 0's of same length. We get 2 runs of 1's when $i = 1, 3, r+1, n-1$. Except for the case where $n = 8, r = 5, i = 7$, $2^i + a$ can never fulfill this condition. We get 3 runs of 1's when $4 \leq i \leq n-2, i \neq r+1$. However we always have 1 run of 1's of length 1 and one run of 1's of length 2 so the above condition is not satisfied. More than 3 runs are impossible in the binary representation of $2^i + a$. For odd n we must have at least 3 runs of 1's of same length in the binary representation of $2^i + a$ which is impossible from the above reasoning. Therefore $|C_z| = n$ and $(2^i + a)2^l \not\equiv (2^i + a), l < n$. Also it is evident from the binary representation of a that $a2^l \not\equiv (2^k + a), k < l$.
3. We know that if two integers b and c belong to the same coset then some cyclic shift of binary representation of b gives c. The binary representation of $2^i + a$ has 1 run of 1's when $i = 0, 2$, however the length of the run is different for $i = 0$ and $i = 2$. We have 2 runs of 1's when $i = 1, 3, r+1, n-1$, however either the length of run's of 1's and 0's is different in each case or the runs of 1's and 0's appear in a different order. We have 3 runs of 1's when $4 \leq i \leq n-2, i \neq r+1$. For each $4 \leq i \leq k$, $2^i + a$ will have a different Hamming weight. For each $k+1 \leq i \leq n-2$, $2^i + a$ has same Hamming weight but either the run's of 1's and 0's are of different length or they appear in different order. Therefore for each $0 \leq i \leq n-1$, $2^i + a$ belongs to a distinct coset. Also $H(a) = H(2^i + a)$ only when $i = 1, k$, however in both cases the length of run's of 1's is different from a.

The 3 necessary and sufficient conditions for $y = x^a$ to have zero bi-affine equations (from Section 6.3.1) are satisfied. □

The proof technique for Theorems 10 and 11 is similar to that of Theorems 8 and 9. So we provide the theorems without proof.

Theorem 10 *Let $n \geq 8$, and $a = 1 + \sum_{i=2}^{r} 2^i = 2^{r+1} - 3, 3 \leq r \leq n-3$. Then power mapping $y = x^a$ over $\mathbb{F}_{2^n}$ has zero bi-affine equations except when n is even and $r = \frac{n}{2} - 1$.*

Theorem 11 *Let $n \geq 8$, and $a = 1 + 2 + 2^2 + \sum_{i=4}^{r} 2^i = 2^{r+1} - 9, 4 \leq r \leq n-3$. Then power mapping $y = x^a$ over $\mathbb{F}_{2^n}$ has zero bi-affine equations except for the following: $(n = 8, r = 5)$, $(n = 9, r = 5, 6)$ and $(n = 10, r = 7)$.*

Several mappings in the constructions given above have good nonlinearity and uniform differential property. For example consider the construction given in Theorem 9. For $n = 8$ power mapping $y = x^{11}$ has nonlinearity 96 and is differentially 10-uniform. For $n = 10$ power mapping $y = x^{11}$ has nonlinearity 480 and is differentially 10-uniform and for $n = 12$ power mapping $y = x^{27}$ has nonlinearity 472 and is differentially 6-uniform. Similarly for construction given in Theorem 10 the mapping $y = x^{13}$ in $\mathbb{F}_{2^{10}}$, has nonlinearity 480 and is differentially 4-uniform. For $n = 12$, mapping $y = x^{1021}$ has nonlinearity 1952 and is differentially 16-uniform.

6.3.2 Power Mappings with Zero Quadratic Equations

If a power mapping is selected randomly then we can use Algorithm 2 given in Section 6.2.2 to calculate the number of quadratic equations it satisfies. Otherwise we provide a construction below to obtain S-box with zero quadratic equations. Consider the power mapping $y = x^a$ on $\mathbb{F}_{2^n}$. We consider the case where $n = 2m$ is even. *The case where n is odd is similar.* From Section 6.2.2, the 3 necessary and sufficient conditions for the power mapping to have zero quadratic equations are:

1.

$$\begin{aligned} H(a) &> 2, \text{ and } H(2^i + a) > 2, \text{ and} \\ H((2^k+1)a) &> 2, \text{ and } H((2^m+1)a) > 2, 0 \le i \le n-1, 1 \le k \le m-1. \end{aligned}$$

2.

$$\begin{aligned} (2^i + a)2^l &\not\equiv (2^i + a) \bmod 2^n - 1, 0 \le i \le n-1, l < n, \text{ and} \\ a2^l &\not\equiv a \bmod 2^n - 1, l < n, \text{ and} \\ (2^k+1)a2^l &\not\equiv (2^k+1)a, \bmod 2^n - 1, 1 \le k \le m-1, l < n, \text{ and} \\ (2^m+1)a2^{l_1} &\not\equiv (2^m+1)a, \bmod 2^n - 1, l_1 < m. \end{aligned}$$

3. Let

$$\mathcal{C} = \{2^i + a : 0 \le i \le n-1\} \cup \{(2^j+1)a : 1 \le j \le m-1\} \cup \{a\} \cup \{(2^m+1)a\}.$$

$\mathcal{C}$ can be a multiset. Then any two elements in $\mathcal{C}$ belong to different cosets modulo $(2^n - 1)$.

Theorem 12 *Let $n \ge 8$, and $a = 1 + 2 + \sum_{i=3}^{r} 2^i, 4 \le r \le n-3$. Then power mapping $y = x^a$ over $\mathbb{F}_{2^n}$ has zero quadratic equations except for the following: $(n = 8, r = 5)$, $(n = 9)$, $(n = 10, r = 5)$ and $(n = 12, r = 4, 8)$.*

Proof. Consider the binary representation of a.

$$a = \overbrace{00\cdots0}^{q}\overbrace{1\cdots11}^{l}011$$

First we consider the case $n = 2m, l < q$. Tables 6.1 and 6.2 show the run distribution (lengths of runs of 1's and 0's) of the binary representation of $(2^i + a), 0 \le i \le n-1$ and $(2^k+1)a, 1 \le k \le m-1$ respectively. For example a run distribution $4, \underline{3}, 1, \underline{2}$ represents the binary form 0000111011 which is 59 in decimal representation. Note that underlined digits represent the length of run of 1's. For

$k = l$ in Table 6.2, there are three different cases for different values of l and for $k = l + 1$ there are 2 different cases for different values of l.

1. From the binary representation of a, Table 6.1 and Table 6.2 it is obvious that $H(a) > 3$, $H(2^i + a) > 2$, $H((2^k + 1)a) > 2$ and $H((2^m + 1)a) > 2, 0 \leq i \leq n - 1, 1 \leq k \leq m - 1$.
2. If $2^i + a, 0 \leq i \leq n - 1 \in C_z$ where $|C_z| < n$ then $|C_z|$ divides n and binary representation of $2^i + a$ is periodic with period $|C_z|$. It is clear from Table 6.1 that the binary representation of $2^i + a$ is never periodic with period less than n. The same reasoning holds for $(2^k + 1)a, 1 \leq k \leq m - 1$ (see Table 6.2). Also from the binary representation of a it is obvious that $|C_a| = n$. Similarly it is easy to check that $(2^m + 1)a2^{l_1} \not\equiv (2^m + 1)a \bmod 2^n - 1, l_1 < m$.
3. If two integers b and c belong to the same coset then some cyclic shift of binary representation of b gives c. This means that both b and c must have same Hamming weight, the number of runs of 1's and 0's must be identical and they must appear in the same order (cyclic). From the binary representations of a and $(2^m + 1)a$, Tables 6.1 and 6.2, it is clear that no two elements of the set $\mathcal{C}$ (see item 3 of Section 6.3.2) belong to the same coset.

For $n = 2m, l \geq q$, the run distribution of $(2^i + a)$ is the same as in Table 6.1. If $k < q$ then the run distribution of $(2^k + 1)a$ is the same as given in Table 6.2. Therefore for $k < q$ items 1, 2 and 3 hold with similar logic. For $k \geq q, (a2^k + a)$ may generate a carry which results in many possible run distributions depending on the actual value of a. Therefore representing the run distribution of $(a2^k + a)$ in a tabular form becomes cumbersome. It is tedious (but not very hard) to check that items 1, 2 and 3 hold for $k \geq q$ as well. Proof for odd n is very similar. □

It can be proved easily that if n is even and $r = n-3$ then power mapping $y = x^a$ in Theorem 12 is bijective if and only if $n \not\equiv 0 \bmod 18$. Also if $r = n-5$ then the above mapping is bijective if and only if $n \not\equiv 0 \bmod 78$. So if $n < \text{lcm}(18, 78) = 234$, then by taking r to be either $n-3$ or $n-5$ we will always get a bijective S-box. Note that bijective mappings exist for many other values of r as well when r is odd. Similar conditions can be found for odd n easily. The power mapping in Theorem 12 is a subset of the power mapping given in Theorem 9 and several mappings with good

Table 6.1: Run distribution in the binary representation of $(2^i + a)$

i	run distribution in$(2^i + a)$	i	run distribution in$(2^i + a)$
0	q,$\underline{l+1}$,2	$l+2$	$q-1$,$\underline{1}$,1,$\underline{l-1}$,1,$\underline{2}$
1	q,$\underline{l+1}$,1,$\underline{1}$	$l+3$	$q-1$,$\underline{l+1}$,1,$\underline{2}$
2	q,$\underline{l+3}$	l+4	$q-2$,$\underline{1}$,1,$\underline{l}$,1,$\underline{2}$
3	$q-1$,$\underline{1}$,$l+1$,$\underline{2}$	$l+5$	$q-3$,$\underline{1}$,2,$\underline{l}$,1,$\underline{2}$
4	$q-1$,$\underline{1}$,$l-1$,$\underline{1}$,1,$\underline{2}$	$l+6$	$q-4$,$\underline{1}$,3,$\underline{l}$,1,$\underline{2}$
		..	.....
5	$q-1$,$\underline{1}$,$l-2$,$\underline{2}$,1,$\underline{2}$	$n-1$	$\underline{1}$,$q-1$,$\underline{l}$,1,$\underline{2}$
..	.....		

Table 6.2: Run distribution in the binary representation of $(2^k + 1)a$

k	run distribution in$(2^k + 1)a$	k	run distribution in$(2^k + 1)a$
1	$q-2$,$\underline{1}$,1,$\underline{l-1}$,3,$\underline{1}$	$l+1$	$q-k$,$\underline{l+2}$,1,$\underline{l-2}$,1,$\underline{2}$ $(l>2)$ $q-3$,$\underline{4}$,2,$\underline{2}$ $(l=2)$
2	$q-3$,$\underline{1}$,2,$\underline{l-2}$,2,$\underline{3}$	$l+2$	$q-k$,$\underline{l+1}$,2,$\underline{l-1}$,1,$\underline{2}$
3	$q-4$,$\underline{1}$,3,$\underline{l-3}$,1,$\underline{1}$,2,$\underline{2}$	$l+3$	$q-k$,$\underline{l}$,1,$\underline{l+2}$,1,$\underline{2}$
4	$q-5$,$\underline{1}$,4,$\underline{l-4}$,1,$\underline{1}$,1,$\underline{1}$,1,$\underline{2}$	$l+4$	$q-k$,$\underline{l}$,1,$\underline{2}$,1,$\underline{l}$,1,$\underline{2}$
5	$q-6$,$\underline{1}$,5,$\underline{l-5}$,1,$\underline{1}$,1,$\underline{2}$,1,$\underline{2}$	$l+5$	$q-k$,$\underline{l}$,1,$\underline{2}$,2,$\underline{l}$,1,$\underline{2}$
6	$q-7$,$\underline{1}$,6,$\underline{l-6}$,1,$\underline{1}$,1,$\underline{3}$,1,$\underline{2}$	l+6	$q-k$,$\underline{l}$,1,$\underline{2}$,3,$\underline{l}$,1,$\underline{2}$
..	.....	..	.....
l	$q-l-1$,$\underline{1}$,$l+1$,$\underline{1}$,1,$\underline{l-3}$,1,$\underline{2}$ $(l>3)$ $q-3$,$\underline{1}$,4,$\underline{3}$ $(l=2)$ $q-4$,$\underline{1}$,4,$\underline{1}$,2,$\underline{2}$ $(l=3)$	$m-1$	$q-k$,$\underline{l}$,1,$\underline{2}$,$m-l-4$,$\underline{l}$,1,$\underline{2}$

nonlinearity and uniform differential property listed for Theorem 9 also belong to the class given in Theorem 12.

6.3.3 Cryptographically Strong Kasami Like Power Mappings

A Kasami exponent [59], $\acute{e}$, is defined as $\acute{e} = 2^{2s} - 2^s + 1$, gcd$(n, s)$=1 and $1 \leq s < \frac{n}{2}$. If we remove the condition gcd(n, s)=1 we can write $e = 2^{2s} - 2^s + 1$, $1 \leq s < \frac{n}{2}$. We will call e, the Kasami like exponent. Suppose $n = 2m$, and we take $s = m - 1$ then we have $e = 2^{2m-2} - 2^{m-1} + 1$ and $a = 2^{m+1}e = 2^{m+1} + 2^{m-1} - 1$. Note that a is the same exponent as in Theorem 8 which has zero bi-affine equations. Based on our extensive experimental results we provide two new conjectures.

Conjecture 1 *Let $n = 2m$, m is even, and $a = 2^{m+1} + 2^{m-1} - 1$. Then mapping $y = x^a$ is maximally nonlinear.*

The validity of Conjecture 1 has been verified up to $n = 24$. Note that since m is even, gcd$(m-1, n) = 1$ and therefore the mapping $y = x^a$ is known to be APN [34]. Also this mapping is provably non-bijective.

Conjecture 2 *Let $n = 2m$, m is odd, and $a = 2^{m+1} + 2^{m-1} - 1$. Then mapping $y = x^a$ is differentially 4-uniform.*

The validity of Conjecture 2 has been verified up to $n = 22$. Note that gcd$(m - 1, n) = 2$ and therefore the mapping $y = x^a$ is known to be maximally nonlinear [33]. Also this mapping is always bijective. Now we provide the following theorem without proof. The proof technique is very similar to the one used in Theorem 12.

Theorem 13 *Let $n = 2m, n \geq 8$ and $a = 2^{m+1} + 2^{m-1} - 1$. Then power mapping $y = x^a$ over $\mathbb{F}_{2^n}$ has $2n$ quadratic equations.*

Note that this mapping has zero bi-affine equations (see Theorem 8), is maximally nonlinear, has high algebraic degree and good uniform differential property. Therefore it is a suitable candidate for cryptographic applications.

Table 6.3: Power mappings with zero bi-affine equations

n	a	$deg(F)$	$nl(F)$	$k-unf.$	n	a	$deg(F)$	$nl(F)$	$k-unf.$
8	11	3	96	10	10	71	4	464	6
	23	4	96	16		79	5	480	4
	29	4	96	10		89	4	472	6
	61	5	96	16		109	5	448	34
10	13	3	480	4		119	6	464	6
	19	3	468	6		125	6	448	34
	23	4	472	6		151	5	464	6
	43	4	464	6		175	6	464	6
	47	5	448	34		191	7	464	6
	53	4	432	34		221	6	448	34
	59	5	464	6		245	6	464	6
	61	5	464	6		251	7	432	34

6.3.4 $\mathcal{AI}$ of Some Well Known Power Mappings

Dobbertin investigated some well known power mappings in [34], i.e., Gold, Dobbertin, Niho, Welch, inverse and Kasami. The number of bi-affine and quadratic equations for some of these power mappings have been found experimentally in [22] (for $n \leq 17$). However using the proof techniques given in Sections 6.3.1 and 6.3.2 the exact number of bi-affine and quadratic equations for the above power mappings can be determined easily. For example Niho exponent e is defined as $e = 2^s + 2^{\frac{s}{2}} - 1$, $n = 2s+1$ when s is even and $e = 2^s + 2^{\frac{3s+1}{2}} - 1$, $n = 2s+1$ when s is odd. It can be easily proved that power mapping $y = x^e$ has zero bi-affine equations for $n \geq 7$ and n quadratic equations for $n > 7$. Since power mapping based on Niho exponent is maximally nonlinear and APN for odd n, it is a good choice for a cryptographically strong S-box in odd number of variables.

The power mapping with Dobbertin exponent although APN is not maximally nonlinear and its algebraic degree is only $\frac{n}{5} + 3$. The algebraic degrees of power mappings with Gold and Welch exponents are 2 and 3 respectively. Therefore these mappings may not be of cryptographic interest for large values of n.

Table 6.4: Power mappings with zero quadratic equations

n	a	$deg(F)$	$nl(F)$	$k-unf.$	n	a	$deg(F)$	$nl(F)$	$k-unf.$
8	27*	4	80	26	10	105*	4	480	10
10	27*	4	472	6		111*	6	432	6
	45*	4	432	6		117*	5	480	6
	51*	4	432	8		123*	6	392	32
	53	4	432	34		183*	6	448	32
	75*	4	480	6		237*	6	480	4
	87*	5	480	4		251	7	432	34

6.4 Experimental Results

In this section we give some cryptographically significant power mappings. Table 6.3 shows all bijective power mappings which have zero bi-affine equations for $n = 8$ and 10. The second column in the table lists the exponents a. Note that a is always the coset leader. The other exponents in the same coset, although not listed in the table, have same properties. The third, fourth and fifth columns show the algebraic degree, nonlinearity and differential uniform property of the mapping respectively. The AES S-box (inverse mapping) has degree 7, nonlinearity 112, and is differentially 4-uniform. From Table 6.3 it is clear that for $n = 8$, no bijective power mapping having zero bi-affine equations is as good as AES S-box. For $n = 10$ the optimal bijective power mapping with zero bilinear equations has exponent 79. This mapping has the same nonlinearity and differential uniform property as the inverse mapping, however there is a tradeoff between the degree and the number of bilinear equations. The inverse mapping has 29 bi-affine equations and its degree is 9 whereas the mapping with exponent 79 has zero bilinear equations and its degree is 5. Note that exponent 79 for $n = 10$ is the same as defined in Theorems 8 and 13. Table 6.4 shows power mappings which have zero quadratic equations for $n = 8$ and 10. An * in the second column indicates that the mapping is non-bijective. From the table it is clear that for $n = 8$ and 10, there is no cryptographically good bijective power mapping with zero quadratic equations.

Table 6.5 shows maximally nonlinear bijective power mappings for $n = 8$ and

Table 6.5: Maximally nonlinear power mappings

n	a	$deg(F)$	$nl(F)$	$k - unf.$	bi-affine eqns.	quadratic eqns.
8	31	5	112	16	16	36
	91	5	112	16	16	36
	127	7	112	4	23	39
10	5	2	480	4	10	40
	13	3	480	4	0	20
	17	2	480	4	15	40
	25	3	480	8	5	10
	41	3	480	8	5	5
	49	3	480	8	5	15
	79	5	480	4	0	20
	107	5	480	8	5	15
	181	5	480	4	15	35
	205	5	480	4	10	40
	511	9	480	4	29	49

10. Columns six and seven in the table give the number of bi-affine and quadratic equations respectively. For $n = 10$, power mappings with exponents 41, 79, and 511 (affine equivalent of inverse exponent) are of cryptographic interest.

Tables 6.3, 6.4 and 6.5 are optimal in terms of bi-affine equations, quadratic equations and nonlinearity respectively. Our experiments show that there does not exist an S-box (for $n \leq 25$) which is optimal in terms of all the properties listed in Table 6.5. Therefore we must relax our criteria to obtain strong S-boxes.

6.5 Cryptographically Significant Power Mappings

To obtain cryptographically strong S-boxes we require that the S-box based on power mapping F must have the following properties:

- Bijective.
- $2^{n-1} - 2^{\frac{n}{2}+1} \leq nl(F) \leq 2^{n-1} - 2^{\frac{n}{2}}$ for n even, and $2^{n-1} - 2^{\frac{n+1}{2}} \leq nl(F) \leq 2^{n-1} - 2^{\frac{n-1}{2}}$ for n odd.

- At most n bi-affine equations.
- At most differentially k-uniform where $k \leq 6$.
- $deg(F) \geq 3$.

For $n = 8$, no S-box satisfies the above criteria. Therefore we have listed an S-box which satisfies all the above properties except that it is differentially 10-uniform. A complete list of all S-boxes (for $7 \leq n \leq 15$) based on power mappings, that satisfy the above criteria, is given in Table 6.6.

Table 6.6: Cryptographically significant mappings

n	a	deg of F	$nl(F)$	diff-k unf	bi-aff eqns	quad eqns
7	11	3	56	2	0	21
	13	3	56	2	0	21
	23	4	56	2	0	21
	27	4	56	2	7	28
	29	4	56	2	0	21
8	11	3	96	10	0	24
	29	4	96	10	0	24
9	13	3	240	2	0	18
	19	3	240	2	0	9
	27	4	240	2	0	9
	45	4	232	4	0	9
	47	5	240	2	0	18
	59	5	240	2	0	18
	87	5	240	2	0	18
	103	5	240	2	9	36
	125	6	232	4	0	9
10	13	3	480	4	0	20
	19	3	464	6	0	10
	23	4	472	6	0	10
	43	4	464	6	0	10
	59	5	464	6	0	10
	61	5	464	6	0	10
	71	4	464	6	0	10
	79	5	480	4	0	20
	89	4	472	6	0	10
	91	5	464	6	5	5
	103	5	464	4	5	5
	115	5	464	6	5	15
	119	6	464	6	0	10
	149	4	464	4	5	5
	151	5	464	6	0	10
	167	5	456	6	5	5
	175	6	464	6	0	10
	191	7	464	6	0	10
	205	5	480	4	10	40
	215	6	464	6	5	5
	223	7	472	4	5	5
	235	6	464	6	5	5
	239	7	456	6	5	5
	245	6	464	6	0	10
	347	6	464	6	5	15
	367	7	472	4	5	5
	379	7	464	6	5	5
11	11	3	960	6	0	22
	13	3	992	2	0	22
	21	3	960	6	11	11
	29	4	960	6	0	11
	35	3	992	2	0	11
	37	3	960	6	0	0
	43	4	992	2	0	22
	47	5	960	6	0	22
	49	3	960	6	0	22
	51	4	960	6	0	0
	53	4	960	6	0	0
11	55	5	960	6	0	0
	57	4	992	2	0	22
	67	3	960	6	11	22
	71	4	960	6	0	11
	73	3	960	6	11	11
	75	4	960	6	0	11
	79	5	960	4	0	11
	81	3	960	6	0	0
	83	4	960	6	0	0
	85	4	960	6	11	11
	95	6	992	2	0	22
	99	4	960	6	11	11
	101	4	968	6	0	0
	103	5	960	6	0	11
	107	5	992	2	0	11
	111	6	968	6	0	0

n	a	deg of F	$nl(F)$	diff-k unf	bi-aff eqns	quad eqns
	113	4	960	6	0	11
	117	5	992	2	0	11
	121	5	960	6	0	22
	125	6	960	6	0	11
	139	4	960	6	0	0
	143	5	992	2	0	22
	149	4	960	6	0	0
	151	5	992	2	0	22
	153	4	960	6	11	22
	157	5	976	6	0	0
	159	6	960	6	0	11
	167	5	968	6	0	0
	171	5	960	6	0	11
	173	5	960	6	0	11
	179	5	968	6	0	0
	181	5	960	6	0	0
	183	6	960	4	0	11
	185	5	960	6	0	0
	187	6	960	6	0	0
	189	6	960	6	0	22
	191	7	960	6	0	11
	201	4	960	6	0	0
	203	5	968	6	0	0
	205	5	960	6	0	11
	213	5	960	6	0	0
	215	6	960	6	0	0
	217	5	960	6	0	0
	219	6	960	6	0	0
	221	6	960	6	0	11
	223	7	968	6	0	0
	229	5	960	6	0	0
	231	6	992	2	11	44
	247	7	960	6	0	0
	249	6	992	2	0	11

n	a	deg of F	$nl(F)$	diff-k unf	bi-aff eqns	quad eqns
	251	7	960	4	0	0
	295	5	960	6	0	0
	301	5	960	6	11	11
11	307	5	960	6	11	22
	309	5	960	6	0	0
	311	6	960	6	0	0
	315	6	992	2	0	22
	317	6	960	6	0	0
	319	7	960	6	0	11
	331	5	968	6	0	0
	333	5	960	6	0	0
	335	6	960	6	0	0
	339	5	976	6	0	0
	343	6	960	6	0	11
	347	6	960	6	0	0
	351	7	960	6	0	0
	359	6	960	6	0	0
	365	6	992	2	11	44
	367	7	960	4	0	0
	373	6	960	6	0	0
	375	7	960	6	0	0
	379	7	960	6	0	11
	381	7	960	6	0	0
	411	6	992	2	11	44
	413	6	992	2	0	22
	423	6	960	6	0	22
	427	6	960	6	0	0
	443	7	960	6	11	11
	463	7	960	4	11	11
	471	7	960	6	0	11
	475	7	960	6	0	0
	477	7	960	6	0	0
	479	8	960	6	0	11
	491	7	968	6	0	0

n	a	deg of F	$nl(F)$	diff-k unf	bi-aff eqns	quad eqns
	493	7	960	6	0	11
	495	8	960	6	11	11
	507	8	968	6	0	0
	687	7	960	6	11	11
	695	7	960	4	0	0
	703	8	960	4	11	11
	727	7	960	6	0	0
	735	8	960	6	0	0
	751	8	960	6	0	0
	763	8	968	6	0	0
	767	9	960	6	11	22
	879	8	960	6	11	11
	959	9	960	6	11	11
12	73	3	1984	4	9	9
	341	5	1952	6	10	10
	731	7	1984	4	9	9
	853	6	1952	6	10	10
13	13	3	4032	2	0	26
	23	4	3968	6	0	13
	35	3	3968	6	0	13
	57	4	4032	2	0	26
	61	5	3968	6	0	13
	67	3	4032	2	0	13
	71	4	4032	2	0	13
13	81	3	3968	6	0	0
	87	5	3968	6	0	0
	107	5	3968	6	0	0
	111	6	3968	6	0	0
	121	5	3968	6	0	13
	133	3	3968	6	0	0
	137	3	3968	6	0	0
	147	4	3968	6	0	0
	151	5	3968	6	0	0
	171	5	4032	2	0	26

n	a	deg of F	$nl(F)$	diff-k unf	bi-aff eqns	quad eqns
	185	5	3968	6	0	0
	191	7	4032	2	0	26
	197	4	3968	6	0	0
	203	5	3968	6	0	0
	205	5	3968	6	0	13
	221	6	3968	6	0	0
	225	4	3968	6	0	13
	229	5	3968	6	0	0
	231	6	3968	6	0	0
	235	6	3968	6	0	0
	241	5	4032	2	0	26
	243	6	3968	6	0	0
	269	4	3968	6	0	0
	275	4	3968	6	0	0
	281	4	3968	6	0	0
	287	6	4032	2	0	26
	291	4	3968	6	0	0
	299	5	3968	6	0	0
	303	6	3968	4	0	0
	307	5	3968	6	0	0
	309	5	3968	6	0	0
	335	6	3968	6	0	0
	339	5	3968	6	0	13
	347	6	4032	2	0	13
	357	5	3968	6	0	0
	365	6	3968	6	0	0
	367	7	4032	2	0	13
	369	5	3968	6	0	0
	375	7	3968	6	0	0
	379	7	3968	6	0	0
	395	5	3968	6	0	0
	401	4	3968	6	0	0
	405	5	3968	6	0	13
	461	6	3968	6	0	0

n	a	deg of F	$nl(F)$	diff-k unf	bi-aff eqns	quad eqns
	465	5	3968	6	0	0
	467	6	3968	6	0	0
	555	5	3968	6	0	0
	581	4	3968	6	0	0
	611	5	3968	6	0	0
	613	5	3968	6	0	0
	617	5	3968	6	0	0
	627	6	3968	6	0	0
	631	7	3968	6	0	0
	635	7	4032	2	0	26
	683	6	3968	6	0	13
13	703	8	3968	6	0	13
	717	6	3968	6	0	0
	723	6	4032	2	0	26
	739	6	3968	6	0	0
	749	7	3968	6	0	13
	763	8	3968	6	0	0
	797	6	3968	6	0	0
	807	6	3968	6	0	0
	809	5	3968	6	0	0
	821	6	3968	6	0	0
	855	7	3968	6	0	0
	861	7	3968	6	0	0
	911	7	4032	2	13	52
	915	6	3968	6	0	0
	919	7	3968	6	0	13
	935	7	3968	6	0	0
	941	7	3968	6	0	13
	947	7	3968	4	0	0
	949	7	3968	6	0	0
	1001	7	3968	6	0	0
	1243	7	4032	2	13	52
	1245	7	4032	2	0	26
	1247	8	3968	6	0	0

n	a	deg of F	$nl(F)$	diff-k unf	bi-aff eqns	quad eqns
	1255	7	3968	6	0	0
	1323	6	3968	6	0	0
	1327	7	3968	6	0	0
	1367	7	3968	6	0	13
	1389	7	3968	6	0	0
	1399	8	3968	6	0	0
	1453	7	4032	2	13	52
	1463	8	3968	6	0	0
	1493	7	3968	6	0	0
	1519	9	3968	6	0	0
	1639	7	4032	2	13	52
	1643	7	3968	6	0	13
	1647	8	3968	6	0	0
	1691	7	4032	2	0	26
	1707	7	3968	6	0	0
	1709	7	3968	6	0	0
	1781	8	3968	6	0	13
	1883	8	3968	6	0	0
	1899	8	3968	6	0	0
	1979	9	3968	6	0	0
	2775	8	3968	6	0	0
14	13	3	8064	4	0	28
	53	4	7936	6	0	0
	71	4	7968	6	0	14
	89	4	8000	6	0	0
	139	4	7936	6	0	0
	173	5	8000	6	0	0
	205	5	8064	4	0	28
	241	5	8064	4	0	28
	319	7	8064	4	0	28
	341	5	8000	6	14	14
	355	5	7984	6	7	7
	371	6	7968	6	7	7
14	437	6	7984	6	0	0

n	a	deg of F	$nl(F)$	diff-k unf	bi-aff eqns	quad eqns
	553	4	7968	6	0	0
	583	5	8000	6	0	0
	593	4	7936	6	0	0
	911	7	8000	6	7	7
	923	7	7968	6	0	14
	937	6	7936	6	0	0
	947	7	8000	6	0	0
	979	7	8064	4	0	28
	1097	4	8000	6	7	7
	1181	6	7968	6	0	0
	1193	5	8000	6	7	7
	1231	7	7968	6	0	0
	1339	7	8064	4	0	28
	1387	7	8000	6	7	7
	1519	9	7968	6	0	0
	1831	7	7936	6	0	14
	1835	7	8000	6	0	14
	1837	7	7984	6	0	0
	1883	8	8000	6	0	0
	1939	7	7968	6	7	7
	1943	8	7968	6	7	7
	2045	10	7968	6	0	14
	2491	8	8000	6	0	14
	2711	7	8000	6	7	7
	2783	9	7936	6	0	0
	2893	7	8064	4	14	56
	2933	8	7968	6	0	0
	2971	8	7992	6	7	7

n	a	deg of F	$nl(F)$	diff-k unf	bi-aff eqns	quad eqns
	3061	9	7968	6	0	0
	3277	7	8064	4	14	56
	3499	8	8000	6	7	7
	3509	8	8000	6	0	0
	3517	9	7968	6	0	14
	3743	9	7968	6	7	7
	3797	8	7936	6	0	0
	4015	10	7984	6	7	7
	4031	11	7992	6	7	7
	5467	8	7936	6	0	14
	6007	10	8000	6	14	14
15	13	3	16256	2	0	30
	73	3	16128	6	15	15
	131	3	16256	2	0	15
	241	5	16256	2	0	30
	383	8	16256	2	0	30
	521	3	16128	6	15	15
	1371	7	16256	2	0	15
	1935	8	16256	2	15	60
	2033	8	16256	2	0	15
	2523	8	16256	2	0	30
	3671	8	16256	2	0	30
	4717	7	16128	6	15	15
	4791	8	16256	2	0	30
	4815	8	16256	2	0	15
	4941	7	16128	6	15	15
	6555	8	16256	2	15	60

6.6 Bijective Power Mappings with Maximum Quadratic Equations

It was stated to be an open problem in [25, 22] to find a bijective nonlinear (n, n) S-box that would give strictly more than $5n$ linearly independent quadratic equations. We provide bijective nonlinear (n, n) S-boxes, where the number of linearly independent quadratic equations is much larger than $5n$. For example from Algorithm 2 it can be checked that for $n = 12$ exponent 683 gives 66 quadratic equations. Also for $n = 14, 16, 18, 20$ and 24 , exponents 2731, 21847, 43691, 174763 and 2796203 give 91, 120, 153, 190 and 276 quadratic equations respectively. Now we give an interesting experimental observation.

Conjecture 3 *Let $y = x^a$ be a bijective nonlinear power mapping over $\mathbb{F}_{2^n}, n \geq 12$ and n is even. Then the maximum number of linearly independent quadratic equations are less than or equal to $\frac{n(n-1)}{2}$.*

The validity of the conjecture has been verified up to $n = 24$ and the upper bound $\frac{n(n-1)}{2}$ is achieved for $n = 12, 14, 16, 18, 20$ and 24. For odd n, $n \geq 5$, our experimental results suggest that this upper bound is $5n$. Also note that for non-bijective power mappings the number of quadratic equations can be much higher than $\frac{n(n-1)}{2}$.

6.7 Conclusions

In this chapter we developed techniques to calculate the number of multivariate bi-affine and quadratic equations for S-boxes based on power mappings. We also provided two simple and efficient algorithms which are practical even for very large S-boxes. The S-box constructions given by us in Sections 6.3.1 and 6.3.2 have provable optimal $\mathcal{AI}$. We also gave experimental results for power S-boxes with high algebraic immunity, nonlinearity, algebraic degree and good uniform differential property. We identified two classes of Kasami like power mappings with good $\mathcal{AI}$. One class is known to be APN and we conjectured that it is also maximally nonlinear. The second class is known to be maximally nonlinear and we conjectured

that it is also differentially 4-uniform. We also identified bijective nonlinear S-boxes that have lowest $\mathcal{AI}$ and conjectured an upper bound on the number of quadratic equations for such S-boxes.

Chapter 7

Equivalent Algebraic Descriptions of S-Boxes

In this chapter we derive equivalent algebraic descriptions of the Inverse and DES S-boxes to study their algebraic structure and other properties. We derive several possible algebraic representations of these S-boxes by using different irreducible polynomials. This study of alternative representation of these S-boxes gives us good insight into their various algebraic properties.

7.1 Background

S-boxes are essential and often the most important building blocks of a symmetric cipher. In most ciphers S-boxes are the only nonlinear components. The S-boxes in a cipher may be all identical as in AES or they may be different from one another as is the case in DES (Data Encryption Standard). Similarly for some ciphers S-boxes are specified explicitly in algebraic terms, for example AES, and for other ciphers simply the input output mapping is provided, i.e., DES. If an S-box is specified in terms of an input/output table, we can derive an algebraic representation through interpolation. We will use the tools from finite fields and sequence design to derive and evaluate the algebraic properties of these S-boxes. The features of our interest are the degree of the algebraic representations, number of monomial terms in their

polynomial and trace representations, and linear complexity of the sequences that correspond to their component functions.

7.2 Equivalent Polynomial Representations of Inverse S-box

The AES S-Box consists of multiplicative inverse in $\mathbb{F}_{2^8}$, defined by the primitive polynomial $x^8+x^4+x^3+x+1$, followed by an affine transformation. S-Boxes used in stream ciphers SNOW1 [37], SNOW2 [38] and SFINKS [14] also use multiplicative inverse to obtain a highly nonlinear transformation. Therefore we will focus on this transformation, i.e., $F : X \rightarrow X^{-1}$ in $\mathbb{F}_{2^8}$ defined by the polynomial $x^8 + x^4 + x^3 + x + 1$. Note that for the rest of this chapter inverse S-Box means the fixed mapping defined above. Recall that a mapping over $\mathbb{F}_{2^n}$ can be represented in the polynomial form as

$$F(x) = \sum_{i=0}^{2^n-1} b_i x^i, b_i \in \mathbb{F}_{2^n}.$$

So we can represent the inverse mapping as $X \rightarrow X^{254 \bmod 255}$. This means that in the polynomial representation coefficient $b_{254} = 1$ and all the remaining coefficients are zero.

Youssef and Gong in [117] showed that for a given input and output of an S-box, changing the defining polynomial of the finite field is equivalent to performing a linear transformation at the input of the S-box. In order to see how this transformation affects the coefficients of the multiplicative inverse function we use Lagrange interpolation (see Section 2.4). To see the effect of changing the defining polynomial we first evaluate the function for all the 256 inputs using the irreducible polynomial used in AES, i.e., $m(x) = x^8 + x^4 + x^3 + x + 1$. Now we interpolate on this output using a different irreducible polynomial $g(x)$. Table 7.1 shows the function calculated by interpolation where $g(x) = x^8+x^7+x^6+x^4+x^3+x^2+1$. The entries correspond to the coefficients in hexadecimal format. For example the coefficient b_{35} is obtained by selecting 3rd row and 5th column in the table which gives 5F.

Table 7.1: Equivalent polynomial representation of Inverse S-Box

$g(x) = x^8 + x^7 + x^6 + x^4 + x^3 + x^2 + 1$																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	FE	F5	E9	41	FD	D9	E7	90	BF	10	EF	8F	4D	AF	47
1	33	FC	78	9B	95	FD	D4	C7	F1	AC	7D	49	EE	11	CC	C1
2	2B	11	E4	36	C8	9	76	B9	AA	E3	EF	13	13	62	EA	C1
3	E8	81	83	65	73	5F	7B	7F	59	15	87	AB	95	71	C1	34
4	BD	F6	F4	A3	E8	E2	98	15	3E	50	AA	D5	88	AC	FA	C0
5	39	A1	36	0	9B	13	9A	F1	8C	23	FD	FB	64	62	FA	C9
6	AE	84	E4	11	2A	46	D3	76	61	27	25	39	5B	78	DF	85
7	4B	9B	18	39	CB	3F	4A	B0	27	F9	6E	93	B8	E7	4B	28
8	EA	92	58	86	6D	10	A9	79	A1	B2	9C	4E	C3	8	9C	75
9	72	BA	49	2E	CE	B7	A	20	DE	2A	5B	B4	67	4D	EA	6A
A	3	89	6D	1	C0	7D	2B	33	B6	37	E6	B2	DF	1B	70	DF
B	90	4A	EA	55	40	9F	4F	F5	DC	D	60	A9	E4	5C	B8	A6
C	33	25	F1	8F	D2	A7	B1	5B	EA	6F	67	5B	62	3D	21	2F
D	70	A2	81	43	51	CF	C6	4D	96	E8	B7	EA	22	A4	B9	BA
E	33	CB	AB	2	B	B8	16	23	74	DC	9A	CD	34	EC	3A	C3
F	ED	2	BD	B6	DC	17	5A	5A	D5	7C	26	6D	9	8E	1	0

In decimal it translates to $b_{53} = 95$. Note that there was only one monomial term in the original representation but after changing the irreducible polynomial all but two monomial terms are nonzero. The representations of the inverse function under all the 28 irreducible polynomials over $\mathbb{F}_2$ of degree 8 were obtained and they all consist of large number of nonzero terms. We have not listed the remaining tables in this thesis due to the lack of space.

7.3 Trace Representation of Inverse S-box

To compute the trace representation of the inverse S-box we first decompose the inverse mapping of the S-box (from $\mathbb{F}_{2^8} \rightarrow \mathbb{F}_{2^8}$) into eight component functions as follows:

$$F(x) = (f_1(x), f_2(x), f_3(x), f_4(x), f_5(x), f_6(x), f_7(x), f_8(x))$$

where each component function is a mapping from $\mathbb{F}_{2^8} \rightarrow \mathbb{F}_2$. Recall that if f is balanced its trace representation given by

Table 7.2: Trace representation of Inverse S-Box

function	Trace Representation
f_1	$Tr_1^8(43x^{127})$
f_2	$Tr_1^8(80x^{127})$
f_3	$Tr_1^8(227x^{127})$
f_4	$Tr_1^8(3x^{127})$
f_5	$Tr_1^8(172x^{127})$
f_6	$Tr_1^8(247x^{127})$
f_7	$Tr_1^8(86x^{127})$
f_8	$Tr_1^8(246x^{127})$

$$f(x) = \sum_{k \in \Gamma(n)} Tr_1^{n_k}(A_k x^k), A_k \in \mathbb{F}_{2^{n_k}}, x \in \mathbb{F}_{2^n}.$$

where $\Gamma(n)$ is the set consisting of all coset leaders modulo $2^n - 1, n_k$ is the size of the coset C_k, and $Tr_1^{n_k}(x)$ is the trace function from $\mathbb{F}_{2^{n_k}} \rightarrow \mathbb{F}_2$.

To compute the trace representation of these functions we use Discrete Fourier Transform (see Section 2.3). The trace representation of each component function under the AES defining polynomial $x^8 + x^4 + x^3 + x + 1$ is shown in Table 7.2. Note that each function has only one trace term.

The trace representation of the component functions obtained by changing the defining polynomial to $x^8 + x^6 + x^5 + x^4 + 1$ is shown in Table 7.3. The first column lists k, the coset leaders modulo 255, and the remaining columns give the corresponding coefficient, A_k, in the trace term for each component function. For example the second row and second column in the table represents the trace term $Tr_1^8(78x)$. The complete trace representation of the function is obtained by the summation of all the trace terms in the given column. The trace representation tables of inverse S-box for all 28 irreducible polynomials of degree 8 over $\mathbb{F}_2$ were obtained and they all consist of large number of trace terms. We have not listed the remaining tables in this thesis due to the lack of space.

Table 7.3: Equivalent trace representation of Inverse S-Box

$g(x) = x^8 + x^6 + x^5 + x^4 + 1$								
k	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
0	0	0	0	0	0	0	0	0
1	78	3A	9	A4	CD	A5	8F	BC
3	4A	22	B6	C7	2E	4E	4B	45
5	64	A	5C	66	2A	40	63	B
7	83	E9	21	D6	84	55	C1	98
9	F8	DC	75	60	4	F2	EB	16
11	DC	23	7A	B4	F5	CC	F0	7B
13	9	DF	6E	C5	BF	6C	89	84
15	ED	10	A7	A2	3E	31	46	53
17	DB	DB	1	97	BD	BD	4C	DA
19	CD	90	72	CA	FA	18	4	86
21	4E	4C	89	32	9	F7	47	4
23	7F	90	B8	E6	47	30	90	A8
25	5A	9	3E	4A	7D	EB	B	D
27	1F	AD	E7	78	9D	7B	70	94
29	50	4E	E4	47	1A	5	6D	F6
31	0	8B	6E	DA	B	2	B5	71
37	62	34	CB	53	53	C4	27	5F
39	6F	36	D9	F6	B2	BF	FB	7A
43	8D	21	24	50	DD	57	8D	EF
45	D3	89	77	47	43	C1	A1	FD
47	48	FB	64	49	3D	F2	FE	71
51	DB	DA	DB	DA	BD	2A	DB	66
53	3	92	2B	F1	F6	66	C6	2B
55	44	2C	47	2D	6C	91	DE	37
59	72	C2	A7	DF	66	48	75	9D
61	D	10	7C	18	20	53	5A	79
63	AE	51	66	77	FE	2D	36	CB
85	DB	DB	DA	DA	0	DB	DB	DA
87	2F	3D	21	E4	74	59	5F	EF
91	0	30	6D	65	B4	84	1E	F2
95	F8	7E	BE	60	6	8B	40	73
111	34	2C	80	8F	FE	57	4A	D9
119	F1	97	BD	DA	1	2A	4D	DB
127	AA	23	BA	3	7E	56	25	57

7.4 Linear Complexity of Inverse Component Sequences

In Section 2.5 we defined the linear complexity of a binary sequence as the size in bits of the shortest LFSR that generates that sequence. Another way to determine the linear complexity of a binary sequence is through its Fourier spectrum using the DFT [52]. Let $\underline{a} = a_i$ be a sequence over $\mathbb{F}_2$ of period N where N divides $2^n - 1$, and let $\underline{A} = A_i$ be its spectral sequence, then the linear complexity of $\underline{a}$ is given by

$$LC(\underline{a}) = |\{k | A_k \neq 0, 0 \leq k \leq N\}|.$$

In other words the linear complexity of the sequence is equal to the Hamming weight of its spectral sequence. From the definition of the trace function it is obvious that we can now determine the linear complexity of a sequence from the trace representation of its corresponding function, i.e.,

$$LC(\underline{a}) = |\{k2^i \bmod 2^n - 1 | A_k \neq 0, k \in \Gamma(n), i = 0, 1, \ldots n_k - 1\}|.$$

If we consider the trace representation of the inverse S-box under the defining polynomial $x^8 + x^4 + x^3 + x + 1$, then each function has only one trace term. This means that the binary sequence corresponding to each function has a linear complexity of 8. However when we change the defining polynomial to $x^8 + x^6 + x^5 + x^4 + 1$, the sequences corresponding to all functions except f_1 and f_5 have linear complexity of 255 which is the maximum value of the linear complexity for a sequence of length $2^8 - 1$.

7.5 Polynomial Representation of DES S-boxes

DES is a 64 bit symmetric block cipher. It has 16 rounds and in each round only half the bits i.e., 32 go through a keyed round function. The round function first expands the 32 bit input to 48 bits followed by the addition of a 48 bit subkey. After the key addition, input goes through 8 distinct S-boxes followed by a permutation

Table 7.4: Polynomial representation of DES S-Box, S1, for $x^6 + x + 1$

F_1	0	1	2	3	4	5	6	7
0	14	62	23	51	4	54	14	31
1	38	17	22	25	4	43	24	11
2	32	33	0	47	49	43	60	30
3	56	58	30	6	0	35	15	10
4	35	56	24	26	29	27	23	6
5	33	50	33	44	25	55	21	62
6	57	53	24	39	61	5	62	21
7	47	2	13	9	41	25	49	0

thus giving the output of the round function. Unlike AES the S-boxes are only described in terms of input/output tables. Each S-box has a 6 bit input and 4 bit output. For more details on DES please refer to [31]. Since the S-boxes of DES are only specified as input/output tables, we can use Lagrange interpolation to find an equivalent polynomial representation over $\mathbb{F}_{2^6}$. Tables 7.4 through 7.11 show the polynomial representation of the eight DES S-boxes (S1 through S8) obtained by using the defining polynomial $x^6 + x + 1$. F_i is the polynomial representation of the S-Box Si. The 64 coefficients in the polynomial representation are arranged in eight rows and eight columns and expressed in decimal format. For example in Table 7.4 second row and first column lists coefficient $b_8 = 38$.

The polynomial representation of S-boxes confirms the high algebraic degree of the S-boxes as well as high number of nonzero monomial terms. Changing the defining polynomial has little effect on the algebraic properties of the S-boxes. The polynomial representation of the 8 S-boxes under all 9 irreducible polynomials of degree 6 over $\mathbb{F}_2$ were obtained and they all consist of a large number of monomials. We have not listed them here due to lack of space.

Table 7.5: Polynomial representation of DES S-Box, S2, for $x^6 + x + 1$

F_2	0	1	2	3	4	5	6	7
0	15	14	38	54	58	8	62	26
1	58	9	26	42	23	29	54	50
2	37	41	46	41	20	30	17	18
3	17	62	27	54	54	15	8	35
4	5	55	56	62	45	29	22	45
5	21	46	21	25	36	2	28	7
6	63	45	62	30	16	51	56	19
7	9	37	47	56	34	57	54	0

Table 7.6: Polynomial representation of DES S-Box, S3, for $x^6 + x + 1$

F_3	0	1	2	3	4	5	6	7
0	10	48	49	17	23	48	41	39
1	4	40	16	38	0	48	15	4
2	22	3	19	62	9	23	54	36
3	43	42	2	19	28	19	53	10
4	6	51	52	13	62	32	17	6
5	18	24	17	14	9	17	46	16
6	34	56	18	57	6	11	3	32
7	1	18	29	13	27	39	16	0

7.6 Trace Representation of DES S-boxes

To compute the trace representation of the DES S-boxes we first decompose the input output mapping of each S-box (from $\mathbb{F}_{2^6} \rightarrow \mathbb{F}_{2^4}$) into 4 functions as follows:

$$F_i(x) = (f_{i1}(x), f_{i2}(x), f_{i3}(x), f_{i4}(x))$$

where i is the index of the S-box and each component function is a mapping from $\mathbb{F}_{2^6} \rightarrow \mathbb{F}_2$. Gong and Golomb in [50] computed the trace representation of all 32 component functions of the 8 DES S-boxes using the primitive polynomial x^6+x+1.

Table 7.7: Polynomial representation of DES S-Box, S4, for $x^6 + x + 1$

F_4	0	1	2	3	4	5	6	7
0	7	1	50	37	23	21	39	58
1	28	2	60	16	47	7	32	61
2	8	4	39	2	2	23	23	24
3	45	57	2	3	55	26	14	31
4	56	41	3	10	34	49	12	26
5	41	33	31	34	9	3	8	44
6	46	40	14	2	45	25	3	61
7	6	46	52	6	10	45	37	0

Table 7.8: Polynomial representation of DES S-Box, S5, for $x^6 + x + 1$

F_5	0	1	2	3	4	5	6	7
0	2	44	52	61	61	48	40	12
1	15	13	57	51	23	62	46	36
2	51	15	17	36	42	20	57	20
3	55	25	13	52	11	25	36	63
4	29	35	54	57	18	45	28	43
5	28	59	18	51	55	37	36	61
6	20	18	9	62	58	35	18	25
7	4	3	62	2	24	41	48	0

In order to see the effect of changing the defining polynomial we compute the trace representations of S-boxes with different irreducible polynomial: $x^6+x^5+x^4+x+1$. Tables 7.12 through 7.19 show the trace representations of eight DES S-boxes. Each entry in the table represents a trace term and the coset leader k corresponding to the trace term is listed in the top row, i.e., in Table 7.12 the third term in the first function, f_{11}, is $Tr_1^6(15x^3)$.

Table 7.9: Polynomial representation of DES S-Box, S6, for $x^6 + x + 1$

F_6	0	1	2	3	4	5	6	7
0	12	21	35	44	24	34	12	57
1	21	13	53	6	18	31	30	9
2	47	24	5	34	56	39	56	50
3	45	36	11	30	13	51	16	53
4	19	52	21	60	0	3	63	57
5	34	53	32	19	14	63	13	38
6	35	15	28	42	50	18	34	61
7	22	31	6	27	26	1	52	0

Table 7.10: Polynomial representation of DES S-Box, S7, for $x^6 + x + 1$

F_7	0	1	2	3	4	5	6	7
0	4	60	51	30	36	15	42	3
1	2	51	8	9	19	2	0	13
2	22	27	58	57	21	31	2	40
3	58	45	27	58	60	9	30	48
4	50	63	44	7	11	27	13	23
5	43	51	20	34	50	48	60	52
6	38	32	3	39	25	21	2	32
7	29	18	40	61	52	28	5	0

7.7 Linear Complexity of DES S-Box Component Sequences

In [50], Gong and Golomb also computed the linear complexity of all the 32 functions associated with the 8 S-boxes using the defining polynomial $x^6 + x + 1$. Their results showed that sequences corresponding to 21 functions had linear span 63 which is the maximal linear span for $\mathbb{F}_{2^6}$ and all the remaining sequences had linear span greater than 57. However when we change the defining polynomial to $x^6 + x^5 + x^4 + x + 1$ the linear complexity of the sequences changes slightly. Table

Table 7.11: Polynomial representation of DES S-Box, S8, for $x^6 + x + 1$

F_8	0	1	2	3	4	5	6	7
0	13	58	6	21	62	58	46	37
1	55	6	6	55	63	7	1	1
2	1	5	48	14	33	62	29	40
3	1	52	16	43	47	50	22	9
4	58	17	51	40	56	25	4	0
5	35	59	50	14	38	38	10	63
6	24	48	46	44	47	10	14	2
7	17	3	31	0	51	7	51	0

Table 7.12: Trace representation of DES S-Box, S1, for $x^6 + x^5 + x^4 + x + 1$

F_1	0	1	3	5	7	9	11	13	15	21	23	27	31
f_{11}	0	32	15	49	15	10	45	12	23	0	62	0	28
f_{12}	1	11	42	24	11	0	35	19	63	1	60	1	40
f_{13}	1	0	0	4	3	37	53	7	52	56	16	11	40
f_{14}	1	53	46	30	59	47	36	9	50	57	28	0	24

Table 7.13: Trace representation of DES S-Box, S2, for $x^6 + x^5 + x^4 + x + 1$

F_2	0	1	3	5	7	9	11	13	15	21	23	27	31
f_{21}	1	16	58	10	33	1	21	40	24	1	6	11	2
f_{22}	1	12	1	19	58	37	26	63	1	56	47	11	54
f_{23}	1	49	45	58	8	47	59	34	54	57	16	1	54
f_{24}	1	33	2	62	55	11	25	43	48	57	38	36	52

7.19 shows that the linear complexity of the sequences corresponding to the functions f_{11},f_{12},f_{13}, and f_{14} are 52, 55, 60 and 61 respectively. Although under this polynomial the linear span of the functions is reduced, this reduction is minimal and in our opinion does not signify a weakness.

Table 7.14: Trace representation of DES S-Box, S3, for $x^6 + x^5 + x^4 + x + 1$

F_3	0	1	3	5	7	9	11	13	15	21	23	27	31
f_{31}	0	18	59	47	2	37	20	2	11	57	32	46	26
f_{32}	1	49	24	14	0	46	24	32	13	56	55	47	42
f_{33}	0	41	10	20	31	11	59	62	49	56	18	1	6
f_{34}	1	51	63	55	4	11	33	50	13	0	4	11	48

Table 7.15: Trace representation of DES S-Box, S4, for $x^6 + x^5 + x^4 + x + 1$

F_4	0	1	3	5	7	9	11	13	15	21	23	27	31
f_{41}	1	7	61	51	56	47	3	47	7	0	61	46	6
f_{42}	1	9	17	29	53	10	3	58	25	1	11	37	6
f_{43}	1	5	5	12	25	37	6	0	42	57	57	47	6
f_{44}	0	53	59	61	3	46	40	30	52	1	15	36	6

Table 7.16: Trace representation of DES S-Box, S5, for $x^6 + x^5 + x^4 + x + 1$

F_5	0	1	3	5	7	9	11	13	15	21	23	27	31
f_{51}	0	3	16	61	29	11	35	16	43	0	23	47	6
f_{52}	1	44	43	21	31	1	47	55	14	1	51	10	40
f_{53}	0	13	21	39	24	36	60	24	7	1	14	47	26
f_{54}	0	19	27	29	3	47	27	54	31	0	50	1	52

Table 7.17: Trace representation of DES S-Box, S6, for $x^6 + x^5 + x^4 + x + 1$

F_6	0	1	3	5	7	9	11	13	15	21	23	27	31
f_{61}	0	23	57	40	37	1	58	25	15	0	31	36	48
f_{62}	0	55	55	45	2	36	45	49	18	56	36	37	54
f_{63}	1	12	29	49	34	10	22	12	49	56	21	37	24
f_{64}	1	11	9	55	14	11	1	45	45	0	3	0	44

Table 7.18: Trace representation of DES S-Box, S7, for $x^6 + x^5 + x^4 + x + 1$

F_7	0	1	3	5	7	9	11	13	15	21	23	27	31
f_{71}	0	20	11	34	25	36	29	40	43	56	21	10	4
f_{72}	0	5	53	23	8	11	8	44	22	0	52	37	28
f_{73}	1	36	63	3	24	47	36	10	26	57	45	47	52
f_{74}	0	62	55	54	22	47	42	58	2	0	61	1	30

Table 7.19: Trace representation of DES S-Box, S8, for $x^6 + x^5 + x^4 + x + 1$

F_8	0	1	3	5	7	9	11	13	15	21	23	27	31
f_{81}	1	9	21	30	28	10	34	0	32	0	11	0	24
f_{82}	0	0	58	41	60	37	52	15	52	0	1	47	54
f_{83}	1	39	51	46	12	11	34	46	7	57	43	0	26
f_{84}	1	26	3	47	1	10	18	56	43	0	12	1	44

7.8 Conclusions

The inverse S-box has a very simple algebraic description under its original defining polynomial. The function is highly nonlinear however it is represented by only one monomial term. Similarly the trace representations of its component functions contain one term each which means a linear span of only 8. How and if this fact can be used in the cryptanalysis of AES is not clear at this time. It is clear however that changing the defining polynomial results in polynomial descriptions that are very complex and trace representations that have large linear spans. This implies that changing the defining polynomial may not be of much significance from algebraic cryptanalysis viewpoint. DES S-boxes on the other hand are only specified in terms of their input/output mapping without any algebraic descriptions. The algebraic representations for all the irreducible defining polynomials have high degrees and large number of monomial terms. Similarly trace representations of the component functions have very large linear spans under all irreducible polynomials. This implies a good design of DES S-box from an algebraic view point.

Chapter 8

Conclusions and Future Work

In this thesis we designed two new stream ciphers and analyzed nonlinear transformations based on power mappings for their resistance against algebraic attacks. In this chapter we provide a summary of our contributions and also list the publications in which these contributions were presented. We also list directions in which this work can be extended and provide some interesting open problems.

8.1 Summary of Contributions

Design and Analysis of WG Ciphers: We began this work by designing a stream cipher suitable for hardware oriented applications. Our goal was to design an efficient and secure stream cipher which produces a keystream with provable randomness properties. We provided the theoretical background to explain our choice of various mathematical transformations used in the design. This was followed by specific guidelines on how to select the parameters and then design a WG cipher for the required security and efficiency. The WG cipher designed for 128-bit security was also submitted to the eSTREAM project in the hardware category. The design and analysis of WG-128 was first presented in [86]. This report is also available at the eSTREAM project webpage [87]. The theoretical background and guidelines on how to design a WG cipher for desired security level are presented in [88].

Design and Analysis of RC4-like Keystream Generator: RC4-like keystream generator has been designed for high speed software applications. Our design was inspired by the simple and efficient design of the RC4 stream cipher. We designed RC4-like keystream generator to address a specific deficiency in the original RC4 design, i.e., its inability to exploit the 32/64 bit architectures of the modern processors. RC4-like keystream generator employs 32/64 bit instructions and it is approximately 3.1 times faster than RC4 on 32-bit machines. We analyzed the security of our design and showed the our design is resistant to several distinguishing attacks that are successful against RC4. We also showed that recovering the internal state of RC4-like keystream generator is much harder than the original RC4 cipher. The RC4-like keystream generator was presented in [53].

Algebraic Immunity of Boolean Functions Based on Power Mappings: Following the design of stream ciphers we turned our attention to nonlinear transformations, i.e., Boolean functions, used in stream ciphers. We investigated the algebraic immunity of the Boolean functions based on power mappings. We developed techniques to analyze the algebraic degree of these functions in the polynomial form. These techniques were then used to derive upper bounds on the algebraic immunity of several well known Boolean functions. We proved that most of these functions have low algebraic immunity. Finally we generalized our results to all functions in the polynomial form. This work was first presented in [82] and was then extended in [83].

Algebraic Immunity of S-Boxes Based on Power Mappings: In this contribution we examined the resistance of S-boxes based on power mappings against algebraic attacks. We developed techniques to find the number of bi-affine and quadratic equations satisfied by an S-box based on power mappings. From these techniques we developed two efficient algorithms to count the total number of these equations for a given S-box. To design secure S-boxes we gave constructions that guarantee zero bi-affine and quadratic equations. Finally we examined these S-boxes for their resistance against linear and differential cryptanalysis and identified several cryptographically strong S-boxes. The efficient algorithms to count the

number of bi-affine and quadratic equations were presented in [84]. An extension of this paper which contains new S-box constructions and the list of cryptographically strong S-boxes is available at [85].

Equivalent Algebraic Descriptions of S-Boxes: In our final contribution we derived equivalent algebraic descriptions of inverse and DES S-boxes. We focused on the polynomial and trace representations of these S-boxes. We computed the number of monomials in these representations and derived the linear complexity of their component functions. This provided us with an insight into their algebraic structure and other cryptographic properties.

8.2 Future Work

In this section we discuss how the work presented in this thesis can be extended in future. We list possible improvements on our results and also list some related open problems.

Design of Stream Ciphers: The WG stream cipher presented in Chapter 3 filters the output of an LFSR. Recently some stream ciphers have been proposed that filter NLFSR or in general they have nonlinear state update functions [36]. These designs are more efficient than the WG ciphers. However it is very difficult to prove that they produce keystream with desired randomness properties. Therefore there is a need to develop techniques to analyze these designs. This will increase our understanding and confidence in such designs. Some open problems related to stream ciphers that use nonlinear state updates are: designing state update functions that guarantee long cycles, calculating the exact period of the keystream, and establishing lower and upper bounds on the linear complexity of the keystream.

It was mentioned in Chapter 4 that RC4-like keystream generator is vulnerable to a distinguishing attack. An obvious next step is to modify RC4-like keystream generator to ensure that it is safe against such distinguishers. We proposed the addition of a few bitwise instructions to break the linearity over $\mathbb{Z}_{2^{32}}$. The final design must be analyzed carefully to ensure that these modifications will make

RC4-like keystream generator safe against distinguishing attacks. The challenge here is to maintain, as much as possible, the simplicity of the present design.

Improving Bounds on Algebraic Immunity of Boolean Functions: In Chapter 5 we used the theory of polynomial functions to prove upper bounds on the algebraic immunity of several well known functions. Experimental results show that in some cases our bounds seem to be tight but in other cases, although very close, our bounds are not tight. A possible extension of this work is to see if these bounds can be improved. More interesting, however, is the problem of finding the lower bounds on the algebraic immunity of these functions. It seems to be a hard problem. Proving an upper bound requires proving the existence of a single function of particular polynomial form whereas proving a lower bound requires proving the absence of all the functions of a particular polynomial form. The latter is a more difficult problem. However this can help us design large functions with provable high algebraic immunity.

Design of Block Ciphers Resistant to Algebraic Attacks: We have identified several cryptographically strong S-boxes in Chapter 6. Some of these S-boxes satisfy very small number of multivariate quadratic equations and their corresponding system of equations is underdefined and very likely more difficult to solve. However an interesting approach is to take an existing block cipher such as AES and replace it's S-box with another one that does not satisfy any multivariate quadratic equation. Unlike the AES S-box, which is optimal in terms of providing resistance against linear and differential cryptanalysis (but very poor against algebraic cryptanalysis), the new S-box will likely decrease the resistance of the block cipher against these two attacks. This decrease in resistance can be quantified and compensated by increasing the number of rounds in the new design. Calculating this increase in the number of rounds and evaluating the efficiency of the new design will provide an interesting alternative. This can also address concerns related to the vulnerability of many block ciphers to algebraic attacks.

Bibliography

[1] G. Agnew, R. Mullin, I. Onyszchuk, and S. Vanstone, An Implementation for a Fast Public-Key Cryptosystem, *Journal of Cryptology*, vol. 3, pp. 63-79, 1991.

[2] F. Armknecht, C. Carlet, P. Gaborit, S. Kuenzli, W. Meier, O. Ruatta, Efficient Computation of Algebraic Immunity for Algebraic and Fast Algebraic Attacks, to appear in *Advances in Cryptology - Eurocrypt 2006*.

[3] F. Armknecht, Algebraic Attacks on Combiners with Memory, *Advances in Cryptology - CRYPTO 2003*, LNCS 2729, pp. 162-176, Springer-Verlag, 2003.

[4] F. Armknecht, Improving Fast Algebraic Attacks *Fast Software Encryption 2004*, LNCS 3017, pp. 65-82, Springer-Verlag, 2003.

[5] F. Armknecht, On the Existence of Low-degree Equations for Algebraic Attacks, *Cryptology ePrint Archive, Report 2004/185*, http://eprint.iacr.org/, 2004.

[6] E. Berlekamp, *Algebraic Coding Theory*, McGraw Hill, New York, 1968.

[7] S. W. Golomb, and G. Gong, *Signal Design for Good Correlation: For Wireless Communication, Cryptography, and Radar*, Cambridge University Press, ISBN 0521821045, 2005.

[8] E. Biham, L. Granboulan, and P. Nguyen. Impossible and Differential Fault Analysis of RC4. *Fast Software Encryption 2005.*

[9] E. Biham, and J. Seberry, Py (Roo) : A Fast and Secure Stream Cipher Using Rolling Arrays, *eSTREAM: The ECRYPT Stream Cipher Project*, Report 2005/023, Available at http://www.ecrypt.eu.org/stream/papers.html

[10] E. Biham, and J. Seberry, Py (Roo) : Pypy: Another Version of Py, *eS-*

TREAM: The ECRYPT Stream Cipher Project, Report 2006/038, Available at http://www.ecrypt.eu.org/stream/papers.html

[11] A. Biryukov and C. D. Canniere, Block Ciphers and Systems of Quadratic Equations, *Fast Software Encryption 2003*, LNCS 2887, pp. 274-289, Springer-Verlag, 2003.

[12] Bluetooth Specification, version 1.1, Available at www.bluetooth.org/spec/.

[13] A. Braeken, J. Lano and B. Preneel, Evaluating the Resistance of Filters and Combiners Against Fast Algebraic Attacks. Eprint on ECRYPT, 2005.

[14] A. Braeken, J. Lano, N. Mentens, B. Preneel and I. Verbauwhede, SFINKS: A Synchronous Stream Cipher for Restricted Hardware Environments, *eSTREAM Project report 2005/026*, Available at http://www.ecrypt.eu.org/stream/.

[15] A. Braeken and B. Preneel, On the Algebraic Immunity of Symmetric Boolean Functions,*Indocrypt 2005*, LNCS 3797, pp.35-48, SPringer-Verlag, 2005.

[16] M. Briceno, I. Goldberg, and D. Wagner, A Pedagogical Implementation of A5/1, http://www.scard.org, May 1999.

[17] F. Chabaud and S. Vaudenay, Links Between Differential and Linear Cryptanalysis,*Advances in Cryptology - Eurocrypt 1994*, LNCS 950, pp. 356-365, Springer-Verlag, 1995.

[18] J. Cheon and D. Lee, Resistance of S-Boxes Against Algebraic Attacks, *Fast Software Encryption 2004*, LNCS 3017, pp. 83-94, Springer-Verlag, 2004.

[19] V. Chepyzhov, T. Johansson, and B. Smeets, A Simple Algorithm for Fast Correlation Attacks on Stream Ciphers, *Fast Software Encryption 2000*, LNCS 1978, pp. 181-195, Springer-Verlag, 2001.

[20] N. Courtois, Fast Algebraic Attacks on Stream Ciphers with Linear Feedback, *Advances in Cryptology-CRYPTO 2003*, LNCS 2729, pp. 176-194, Springer-Verlag, 2003.

[21] N. Courtois, Algebraic Attacks on Combiners with Memory and Several Outputs, *ICISC 2004*, LNCS 3506, pp. 3-20, Springer-Verlag, 2004.

[22] N. Courtois, B. Debraize and E. Garrido, On Exact Algebraic [Non]Immunity

of S-boxes Based on Power Functions, *Cryptology ePrint Archive, Report 2005/203*, http://eprint.iacr.org/, 2005.

[23] N. Courtois and W. Meier, Algebraic Attacks on Stream Ciphers with Linear Feedback, *Advances in Cryptology - Eurocrypt 2003*, LNCS 2656, pp. 346-359, Springer-Verlag, 2003.

[24] N. Courtois and W. Meier, Algebraic Attacks on Stream Ciphers with Linear Feedback, Extended version of [23], available at http://cryptosystem.net/stream

[25] N. Courtois and Pieprzyk J., Cryptanalysis of Block Ciphers with Overdefined Systems of Equations, *Advances in Cryptology - Asiacrypt 2002*, LNCS 2501. Springer-Verlag, 2002.

[26] J. Daemen, and V.Rijmen, *The Design of Rijndael*, Springer-Verlag, 2002.

[27] D. K. Dalai, K. C. Gupta and S. Maitra. Notion of Algebraic Immunity and Its Evaluation Related to Fast Algebraic Attacks, *Second International Workshop, Boolean Function: Cryptography and Applications 2006 (BFCA 06).*

[28] D. K. Dalai, K. C. Gupta and S. Maitra, Cryptographically Significant Boolean Functions: Construction and Analysis in Terms of Algebraic Immunity. *Fast Software Encryption 2005*, LNCS 3557, pp. 98-111, Springer-Verlag, 2005.

[29] D. K. Dalai, S. Maitra and S. Sarkar, Basic Theory in Construction of Boolean Functions with Maximum Possible Annihilator Immunity. To appear in *Designs, Codes and Cryptography.*

[30] E. Dawson, A. Clark, J. Golic, W. Millan, L. Penna, and L. Simpson, The LILI-128 Keystream Generator, *Proceedings of First NESSIE Workshop*, Heverlee, Belgium, 2000.

[31] Data Encryption Standard (DES), emphFIPS PUB-43 U.S. Department of Commerce/National Institute of Standards and Technology. Available at http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[32] J. Dillon, and H. Dobbertin, New Cyclic Difference Sets with Singer Parameters, *Finite Fields and Their Application*, 10(2004), pp. 342-389.

[33] H. Dobbertin, One-to-One Highly Nonlinear Power Functions on $GF(2^n)$. *Ap-*

plicable Algebra in Engineering, Communication and Computing, Vol. 9, pp. 139-152, 1998.

[34] H. Dobbertin, Almost Perfect Nonlinear Power Functions on $GF(2^n)$: The Welch Case. *IEEE Transactions on Information Theory*, Vol. 45, No. 4, pp. 1271-1275, 1999.

[35] H. Dobbertin, Almost Perfect Nonlinear Power Functions on $GF(2^n)$: The Niho Case. *Information and Computation*, Vol. 151, pp. 57-72, 1998.

[36] eSTREAM - The ECRYPT Stream Cipher Project, *http://www.ecrypt.eu.org/stream/*

[37] P. Ekdahl, and T. Johansson, SNOW-A New Stream Cipher, *Proceedings of First NESSIE Workshop*, Heverlee, Belgium, 2000.

[38] P. Ekdahl, and T. Johansson, SNOW-A New Version of the Stream Cipher SNOW, *Selected Areas in Cryptography, 2002*, LNCS 2595, pp. 47-61, Springer-Verlag 2003.

[39] J. C. Faugère, A New Efficinet Algorithm for Computing Grobner Basis (F_4), *Journal of Pure and Applied Algebra*, Vol. 139, pp. 61-88, 1999.

[40] J. C. Faugère, A New Efficinet Algorithm for Computing Grobner Basis without Reduction to Zero (F_5), *International Symposium on Symbolic and Algebraic Computation*, ACM Special Interest Group on Journal of Pure and Applied Algebra, available at http://www-calfor.lip6.fr/ jcf/Papers/@papers/f5.pdf

[41] Gui-Liang Feng, A VLSI Architecture for Fast Inversion in GF(2^m), *IEEE Transactions on Computer*, vol. 38, No. 10, pp. 1383-1386, October 1989.

[42] H. Finney, An RC4 cycle that can't happen, *Post in sci.crypt, September 1994.*

[43] P. Flajolet and A. M. Odlyzko. Random Mapping Statistics (Invited), *Eurocrypt '89*, vol. 434 of LNCS, pp. 329-354, Springer-Verlag, 1990.

[44] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. *SAC 2001* Vol. LNCS 2259, pp. 1-24, Springer-Verlag, 2001.

[45] S. Fluhrer and D. McGrew. Statistical Analysis of the Alleged RC4 Keystream Generator. *Fast Software Encryption 2000* Vol. LNCS 1978, pp. 19-30, Springer-Verlag, 2000.

[46] R. Gold, Maximal Recursive Sequences with 3 valued cross-correlation function, *IEEE Transactions on Information Theory*, Vol. 14, pp. 154-156, 1968.

[47] J. Golić. Linear Statistical Weakness of Alleged RC4 Keystream Generator, *Eurocrypt '97*, vol. 1233 of LNCS, pp. 226-238, Springer-Verlag, 1997.

[48] J. Dj. Golić. Iterative Probabilistic Cryptanalysis of RC4 Keystream Generator, *ACISP'2000*, Vol. 1841 of LNCS, pages 220–233. Springer-Verlag, 2000.

[49] J. Golić, V. Bagini, and G. Morgari, Linear Cryptanalysis of Bluetooth Stream Cipher, *EUROCRYPT-2002*, LNCS 2332, pp. 238-255, Springer-Verlag 2002.

[50] G. Gong, and S. W. Golomb, Transform domain analysis of DES, *IEEE Transactions on Information Theory*, vol. 45, No.6, pp. 2065-2073, Sep. 1999.

[51] S. W. Golomb and G. Gong, Hyper-Cyclotomic Algebra, *Sequences and their Applications, SETA'01*, Discrete Mathematics and Theoretical Computer Science, Springer, 2001, pp. 154-165. CORR 2001-33.

[52] S. W. Golomb, and G. Gong, *Signal Design for Good Correlation: For Wireless Communication, Cryptography, and Radar*, Cambridge University Press, ISBN 0521821045, 2005.

[53] G. Gong, K. C. Gupta, M. Hell and Y. Nawaz, Towards a General RC4-like Keystream Generator, *Information Security and Cryptology, First SKOLIS Conference*, Vol. 3822 of LNCS, pp. 162-174, Springer Verlag, 2005.

[54] G. Gong, and A. Youssef, Cryptographic Properties of the Welch-Gong Transformation Sequence Generators, *IEEE Transactions on Information Theory*, vol. 48, No. 11, pp. 2837-2846, Nov. 2002.

[55] G.Gong, On Existence and Invariant of Algebraic Attakcs, *Technical report CORR2004-16, Centre for Applied Cryptographic Research, University of Waterloo*, Available at http://www.cacr.math.uwaterloo.ca/

[56] A. Grosul and D. Wallach. A related key cryptanalysis of RC4. *Department of Computer Science, Rice University, Technical Report TR-00-358*, June 2000.

[57] P. Hawkes, G. Rose, Rewriting Variables: The Complexity of Fast Algebraic Attacks on Stream Ciphers, *Advances in Cryptology - Crypto 2004*, volume LNCS 3152, pp. 390-406, Springer-Verlag, 2004.

[58] R. Jenkins. Isaac and RC4, Available at http://burtleburtle.net/bob/rand/isaac.html.

[59] T. Kasami, The Weight Enumerators for Several Classes of Subcodes of the Second Order Binary Reed-Muller Codes, *Infor. Contr.*, Vol. 18, pp. 369-394, 1971.

[60] N. Keller, S. Miller, I. Mironov, and R. Venkatesan, MV3: A New Word Based Stream Cipher Using Rapid Mixing and Revolving Buffers, *CT-RSA 2007*, LNCS 4377, pp. 1-19, Springer-Verlag 2007.

[61] A. Klapper, and M. Goresky, Feedback Shift Registers, 2-Adic Span and Combiners with Memory, *Journal of Cryptology*, Vol. 10, pp. 111-147, 1997.

[62] A. Klapper, and J. Xu, Algebraic Feedback Shift Registers, *Theoretical Computer Science*, Vol. 226, pp. 61-93, 1999.

[63] A. Klapper, Distribution Properties of d-FCSR Sequences, *Journal of Complexity*, Vol. 20, pp. 305-317, 2004.

[64] L. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege. Analysis Methods for (Alleged) RC4. *Asiacrypt '98*, vol. 1514 of LNCS, pp. 327-341, Springer-Verlag, 1998.

[65] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and their Applications*. Cambridge University Press, 1994.

[66] M. D. MacLaren and G Marsaglia. Uniform random number generation. *J. ACM*, vol. 15, pp. 83–89, 1965.

[67] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*. North Holland, 1986.

[68] I. Mantin. Predicting and Distinguishing Attacks on RC4 Keystream Generator. *Eurocrypt 2005*, Vol. 3494 of LNCS, pp. 491-506, Springer-Verlag, 2005.

[69] I. Mantin and A. Shamir. A Practical Attack on Broadcast RC4. *Fast Software Encryption 2001*, Vol. 2355 of LNCS, pp. 152-164, Springer-Verlag, 2001.

[70] I. Mantin. The Security of the Stream Cipher RC4. *Master Thesis (2001) The Weizmann Institute of Science*

[71] L. Massey, and J. Omura, Computational Method and Apparatus for Finite Field Arithmetic, *US Patent No. 4,587,627*, 1986.

[72] J. Massey, Shift-Register Synthesis and BCH Decoding, *IEEE Transactions on Information Theory*, Vol. 15, No. 1, pp. 122-127, 1969.

[73] A. Maximov, T. Johansson, and S. Babbage, An Improved Correlation Attack on A5/1, *Selected Areas in Cryptography 2004*,LNCS 3357, pp. 1-18, Springer-Verlag, 2004.

[74] A. Maximov. Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of the RC4 Family of Stream Ciphers. *Fast Software Encryption 2005.*

[75] W. Meier, E. Pasalic, and C. Carlet, Algebraic Attacks and Decomposition of Boolean Functions, *Advances in Cryptology EUROCRYPT-2004*, LNCS 3027, pp.474-491, Springer-Verlag, 2004.

[76] A. Menzes, P.Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[77] I. Mironov. Not (So) Random Shuffle of RC4. *Crypto2002*, Vol. 2442 of LNCS, pp. 304-319, Springer-Verlag, 2002.

[78] S. Mister and S. Tavares. Cryptanalysis of RC4-like Ciphers. *SAC '98*, vol. 1556 of LNCS, pp. 131-143, Springer-Verlag, 1999.

[79] R. Mullin, I. Onyszchuk, and S. Vanstone, Optimal Normal Bases in GF(p^n), *Discrete Applied Mathematics*, vol. 22, pp. 149-161, 1989.

[80] S. Murphy and M. Robshaw, Essential Algebraic Structure within AES, *Advances in Cryptology - Crypto 2002*, LNCS 2442, pp.1-16, Springer-Verlag, 2002.

[81] S. Murphy and M. Robshaw, Comments on the Security of the AES and the XSL Technique, *Electronic Letters*, Vol. 39, pp. 26-38, 2003.

[82] Y. Nawaz, G. Gong and K. C. Gupta, Upper Bounds on Algebraic Immunity of Boolean Power Functions, *Fast Software Encryption 2006*, Vol. 4047 of LNCS, pp. 375-389, Springer Verlag, 2006.

[83] Y. Nawaz, G. Gong and K. C. Gupta, Upper Bounds on Algebraic Immunity of Boolean Power Functions, (*Preprint*)

[84] Y. Nawaz, K. C. Gupta, and G. Gong, Efficient Techniques to Find Alge-

braic Immunity of S-boxes Based on Power Mappings, *Workshop on Coding and Cryptography (WCC)*, Versailles, France, April 2007.

[85] Y. Nawaz, K. C. Gupta, and G. Gong, Algebraic Immunity of S-Boxes Based on Power Mappings: Analysis and Construction, *Cryptology ePrint Archive, Report 2006/322*, http://eprint.iacr.org/, 2006.

[86] Y. Nawaz and Guang Gong, The WG Stream Cipher, *Workshop on Symmetric Key Encryption*, Aarhus, Denmark, May 26- May 27, 2005.

[87] Y. Nawaz and Guang Gong, The WG Stream Cipher, *eSTREAM: The ECRYPT Stream Cipher Project*, Report 2005/033, Available at http://www.ecrypt.eu.org/stream/papers.html

[88] Y. Nawaz and Guang Gong, WG: A Family of Stream Ciphers with Designed Randomness Properties, (*Preprint*)

[89] European project IST-1999-12324 on New European Schemes for Signature, Integrity, and Encryption. *www.cryptonessie.org/*

[90] NESSIE partners, Performance of Optimized Impleentations of the NESSIE Primitives, Technical Report NES/DOC/TEC/WP6/D21/2,2003. Available at http://www.nessie.eu.org/nessie.

[91] Y. Niho, Multi-valued Cross-correlation Functions Between Two Maximal Linear Recursive Sequences, *Ph. D. dissertation*, Dept. Elec. Eng., Univ. Southern California (USCEE Rep. 409), 1972.

[92] P. Ning, and Y. Yin, Efficient Software Implementation for Finite Field Multiplication in Normal Basis, *ICICS 2001*, LNCS 2229, pp. 177-188, Springer-Verlag, 2001.

[93] NIST statistical tests suite with documentation, Available at http://stat.fsu.edu/∼geo/diehard.html.

[94] J. S. No, S. W. Golomb, G. Gong, H. K. Lee, and P. Gaal, Binary Pseudorandom Sequences of period $2^n - 1$ with Ideal Correlation Properties, *IEEE Transactions on Information Theory*, Vol. 44, No. 2, pp. 814-817, March 1998.

[95] K. Nyberg, Differentially Uniform Mappings for Cryptography, *Advances in Cryptology - Eurocrypt 1993*, LNCS 765, pp.55-64, Springer-Verlag, 1994.

[96] C. Paar, Efficinet VLSI Architectures for Bit Parallel Computation in Galois Fields, *Doctoral dissertation, Institute for Experimental Mathemetics, University of Essen*, Germany, 1994.

[97] S. Paul and B. Preneel. Analysis of Non-fortuitous Predictive States of the RC4 Keystream Generator. *Indocrypt 2003*, vol. 2904 of LNCS, pp. 52-67, Springer-Verlag, 2003.

[98] S. Paul and B. Preneel. A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. *Fast Software Encryption 2004*, Vol. 3017 of LNCS, pp. 245-259, Springer-Verlag, 2004.

[99] S. Paul and B. Preneel, Distinguishing Attack on Stream Cipher Py, *Fast Software Encryption 2006*, Vol. 4047 of LNCS, pp. 405-421, Springer Verlag, 2006.

[100] S. Paul and B. Preneel, On the (In)security of Stream Ciphers Based on Arrays and Modular Addition. *Advances in Cryptology - ASIACRYPT 2006*, Vol. 4284 of LNCS, pp. 69-83, Springer-Verlag, 2006.

[101] M. Pudovkina. Statistical Weaknesses in the Alleged RC4 keystream generator. *Cryptology ePrint Archive 2002-171, IACR, 2002.*

[102] A. Reyhani-Masoleh, and A. Hassan, A New Construction of Massey-Omura Parallel Multiplier over GF(2^m), *IEEE Transactions on Computers*, vol. 51, No. 5, pp. 511-520, May 2002.

[103] A. Reyhani-Masoleh, and A. Hassan, Efficient Digit-Serial Normal Basis Multipliers over GF(2^m), *ACM Transactions on Embedded Computer Systems*, special issue on embedded systems and security, vol. 3, No. 3, pp. 575-592, Aug. 2004.

[104] A. Reyhani-Masoleh, and A. Hassan, Low Complexity Word-Level Sequential Normal Basis Multiplier, *IEEE Transaction on Computers*, vol. 54, N0. 2, Feb. 2005.

[105] A. Reyhani-Masoleh, and A. Hassan, Fast Normal Basis Multiplication Using General Purpose Processors, *IEEE Transaction on Computers*, vol. 52, N0. 11, Nov. 2003.

[106] R. Rivest, The RC4 Encryption Algorithm, *RSA Data Security, Inc.*, Mar. 1992.

[107] A. Roos. Class of weak keys in the RC4 stream cipher. *Post in sci.crypt, September 1995.*

[108] I. Schaumuller-Bichl, Cryptanalysis of the Data Encryption Standard by the Method of Formal Coding, *Advances in Cryptology - Eurocrypt 1982*, LNCS 149, pp.235-255, Springer-Verlag, 1983.

[109] G. J. Simmons et al. Contemproary Cryptography: The Science of Information Integrity *IEEE Press, New York, USA* ISBN 0-87942-277-7

[110] A. Stubblefield, J. Ioannidis, and A. Rubin. Using the Fluhrer, Mantin and Shamir attack to break WEP. *Proceedings of the 2002 Network and Distributed Systems Security Symposium*, pp. 17-22, 2002.

[111] B. Sunar, and C. Koc, An Efficient Optimal Normal Basis Type II Multiplier, *IEEE Transactions on Computers*, vol. 50, No. 1, pp. 83-88, Jan. 2001.

[112] Y. Tsunoo, T. Saito, H. Kubo, M. Shigeri, T. Suzaki, and T. Kawabata. The Most Efficient Distinguishing Attack on VMPC and RC4A. *SKEW 2005.*

[113] C. Wang, T. Troung, H. Shao, L. Deutsch, J. Omura, and I. Reed, VLSI Architecture for Computing Multiplications and Inverses in GF(2^m), *IEEE Transactions on Computers*, vol. 34, No. 8, pp. 709-716, Aug. 1985.

[114] D. Watanabe, S. Furuya, H. Yoshida, and B. Preneel, A New Keystream Generator MUGI, *Fast Software Encryption 2002*, LNCS 2365, pp. 179-194, Springer-Verlag, 2002.

[115] H. Wu, Stream Cipher HC-256, *eSTREAM: The ECRYPT Stream Cipher Project*, Report 2005/011, Available at http://www.ecrypt.eu.org/stream/papers.html

[116] A.M.Yousef and G.Gong, Hyper-bent Functions, *Advances in Cryptology, EUROCRYPT-2001*, volume LNCS 2045, pp.406-419. Springer-Verlag, 2001

[117] A. Youssef, and G. Gong, On the Interpolation Attacks on Block Ciphers, *Fast Software Encryption 2000*, LNCS 1978, pp. 109-120, Springer-Verlag 2001.

[118] B. Zoltak. VMPC One-Way Function and Stream Cipher. *Fast Software Encryption 2004*, vol. 3017 of LNCS, pp. 210-225, Springer-Verlag, 2004.

Appendix A

C_N Matrix for Normal Basis Multiplier in $\mathbb{F}_{2^{29}}$

$$\begin{pmatrix}
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1\\
0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1
\end{pmatrix}$$

C Implementation of a Normal Basis Multiplier in $\mathbb{F}_{2^{29}}$

Following is the C implementation (32 bit word size) of the normal basis multiplier for the normal basis defined by γ in Section 3.5. The implementation is based on the algorithm given in [92].

```
#define ROTL29(v, n)
(unsigned)(((v) ≪ (n)) | ((v) ≫ (29 - (n))))& 0xFFFFFFF8

void mult(unsigned int a, unsigned int b, unsigned int* c){

    unsigned int A[29], B[29];

    /*precomputation*/
    A[0]=a & 0xFFFFFFF8; B[0]=b & 0xFFFFFFF8;
    A[1]=ROTL29(A[0], 1); B[1]=ROTL29(B[0],1);
    A[2]=ROTL29(A[0], 2); B[2]=ROTL29(B[0],2);
    A[3]=ROTL29(A[0], 3); B[3]=ROTL29(B[0],3);
    A[4]=ROTL29(A[0], 4); B[4]=ROTL29(B[0],4);
    A[5]=ROTL29(A[0], 5); B[5]=ROTL29(B[0],5);
    A[6]=ROTL29(A[0], 6); B[6]=ROTL29(B[0],6);
    A[7]=ROTL29(A[0], 7); B[7]=ROTL29(B[0],7);
    A[8]=ROTL29(A[0], 8); B[8]=ROTL29(B[0],8);
    A[9]=ROTL29(A[0], 9); B[9]=ROTL29(B[0],9);
    A[10]=ROTL29(A[0], 10); B[10]=ROTL29(B[0],10);
    A[11]=ROTL29(A[0], 11); B[11]=ROTL29(B[0],11);
    A[12]=ROTL29(A[0], 12); B[12]=ROTL29(B[0],12);
    A[13]=ROTL29(A[0], 13); B[13]=ROTL29(B[0],13);
    A[14]=ROTL29(A[0], 14); B[14]=ROTL29(B[0],14);
    A[15]=ROTL29(A[0], 15); B[15]=ROTL29(B[0],15);
    A[16]=ROTL29(A[0], 16); B[16]=ROTL29(B[0],16);
    A[17]=ROTL29(A[0], 17); B[17]=ROTL29(B[0],17);
    A[18]=ROTL29(A[0], 18); B[18]=ROTL29(B[0],18);
    A[19]=ROTL29(A[0], 19); B[19]=ROTL29(B[0],19);
    A[20]=ROTL29(A[0], 20); B[20]=ROTL29(B[0],20);
```

```
A[21]=ROTL29(A[0], 21); B[21]=ROTL29(B[0],21);
A[22]=ROTL29(A[0], 22); B[22]=ROTL29(B[0],22);
A[23]=ROTL29(A[0], 23); B[23]=ROTL29(B[0],23);
A[24]=ROTL29(A[0], 24); B[24]=ROTL29(B[0],24);
A[25]=ROTL29(A[0], 25); B[25]=ROTL29(B[0],25);
A[26]=ROTL29(A[0], 26); B[26]=ROTL29(B[0],26);
A[27]=ROTL29(A[0], 27); B[27]=ROTL29(B[0],27);
A[28]=ROTL29(A[0], 28); B[28]=ROTL29(B[0],28);

/* multiplication */
*c=A[0] & B[1];
*c^= A[1] & (B[0] ^B[21]);
*c^= A[2] & (B[6] ^B[21]);
*c^= A[3] & (B[13] ^B[18]);
*c^= A[4] & (B[11] ^B[27]);
*c^= A[5] & (B[17] ^B[20]);
*c^= A[6] & (B[2] ^B[22]);
*c^= A[7] & (B[13] ^B[25]);
*c^= A[8] & (B[9] ^B[10]);
*c^= A[9] & (B[8] ^B[14]);
*c^= A[10] & (B[8] ^B[26]);
*c^= A[11] & (B[4] ^B[14]);
*c^= A[12] & (B[17] ^B[24]);
*c^= A[13] & (B[3] ^B[7]);
*c^= A[14] & (B[9] ^B[11]);
*c^= A[15] & (B[24] ^B[26]);
*c^= A[16] & (B[19] ^B[23]);
*c^= A[17] & (B[5] ^B[12]);
*c^= A[18] & (B[3] ^B[22]);
*c^= A[19] & (B[16] ^B[27]);
*c^= A[20] & (B[5] ^B[28]);
*c^= A[21] & (B[1] ^B[2]);
*c^= A[22] & (B[6] ^B[18]);
```

```
    *c^= A[23] & (B[16] ^B[25]);
    *c^= A[24] & (B[12] ^B[15]);
    *c^= A[25] & (B[7] ^B[23]);
    *c^= A[26] & (B[10] ^B[15]);
    *c^= A[27] & (B[4] ^B[19]);
    *c^= A[28] & (B[20] ^B[28]);
}
```

C Implementation of a Normal Basis Inverter in $\mathbb{F}_{2^{29}}$

The following C implementation (32 bit word size) uses the normal basis multiplication to find inverse of an element in $F_{2^{29}}$. The following implementation is based on the algorithm given in [41].

```
#define ROTL29(v, n)
(unsigned)(((v) ≪ (n)) | ((v) ≫ (29 - (n))))& 0xFFFFFFF8
#define ROTR29(v, n) ROTL29(v, 29 - (n))


void inverse(unsigned int a, unsigned int* b){
    b=a
    /*4*/
    b=ROTL29(b, 16);
    mult(b, ROTR29(b, 8), b);
    mult(b, a, b);
    /*3*/
    b=ROTL29(b, 8);
    mult(b, ROTR29(b, 4), b);
    mult(b, a, b);
    /*2*/
    b=ROTL29(b, 4);
    mult(b, ROTR29(b, 2), b);
    /*1*/
    mult(b, ROTR29(b, 1), b);
}
```

Appendix B

Initial Values for RC4(8,32)

Initial values for RC4$(8, 32)$ in hexadecimal format.

a_0 = 144D4800	a_1 = 32736901	a_2 = 51988B02	a_3 = 6FBEAD03
a_4 = 8DE4CE04	a_5 = AC0AF005	a_6 = CA301206	a_7 = E8553407
a_8 = 067B5508	a_9 = 25A17709	a_{10} = 43C7990A	a_{11} = 61ECBA0B
a_{12} = 7F12DC0C	a_{13} = 9E38FE0D	a_{14} = BC5E1F0E	a_{15} = DA84410F
a_{16} = F9A96310	a_{17} = 17CF8411	a_{18} = 35F5A612	a_{19} = 531BC813
a_{20} = 7240E914	a_{21} = 90660B15	a_{22} = AE8C2D16	a_{23} = CCB24E17
a_{24} = EBD87018	a_{25} = 09FD9219	a_{26} = 2723B31A	a_{27} = 4649D51B
a_{28} = 646FF71C	a_{29} = 8294181D	a_{30} = A0BA3A1E	a_{31} = BFE05C1F
a_{32} = DD067D20	a_{33} = FB2C9F21	a_{34} = 1951C122	a_{35} = 3877E223
a_{36} = 569D0424	a_{37} = 74C32625	a_{38} = 93E84726	a_{39} = B10E6927
a_{40} = CF348B28	a_{41} = ED5AAC29	a_{42} = 0C80CE2A	a_{43} = 2AA5F02B
a_{44} = 48CB112C	a_{45} = 66F1332D	a_{46} = 8517552E	a_{47} = A33C762F
a_{48} = C1629830	a_{49} = E088BA31	a_{50} = FEAEDB32	a_{51} = 1CD4FD33
a_{52} = 3AF91F34	a_{53} = 591F4035	a_{54} = 77456236	a_{55} = 956B8437
a_{56} = B490A538	a_{57} = D2B6C739	a_{58} = F0DCE93A	a_{59} = 0E020A3B
a_{60} = 2D282C3C	a_{61} = 4B4D4E3D	a_{62} = 6973703E	a_{63} = 8799913F
a_{64} = A6BFB340	a_{65} = C4E4D541	a_{66} = E20AF642	a_{67} = 01301843
a_{68} = 1F563A44	a_{69} = 3D7C5B45	a_{70} = 5BA17D46	a_{71} = 7AC79F47
a_{72} = 98EDC048	a_{73} = B613E249	a_{74} = D438044A	a_{75} = F35E254B
a_{76} = 1184474C	a_{77} = 2FAA694D	a_{78} = 4ED08A4E	a_{79} = 6CF5AC4F
a_{80} = 8A1BCE50	a_{81} = A841EF51	a_{82} = C7671152	a_{83} = E58C3353
a_{84} = 03B25454	a_{85} = 21D87655	a_{86} = 40FE9856	a_{87} = 5E24B957
a_{88} = 7C49DB58	a_{89} = 9B6FFD59	a_{90} = B9951E5A	a_{91} = D7BB405B
a_{92} = F5E0625C	a_{93} = 1406835D	a_{94} = 322CA55E	a_{95} = 5052C75F
a_{96} = 6F78E860	a_{97} = 8D9D0A61	a_{98} = ABC32C62	a_{99} = C9E94D63
a_{100} = E80F6F64	a_{101} = 06349165	a_{102} = 245AB266	a_{103} = 4280D467
a_{104} = 61A6F668	a_{105} = 7FCC1769	a_{106} = 9DF1396A	a_{107} = BC175B6B
a_{108} = DA3D7C6C	a_{109} = F8639E6D	a_{110} = 1688C06E	a_{111} = 35AEE16F
a_{112} = 53D40370	a_{113} = 71FA2571	a_{114} = 8F204772	a_{115} = AE456873
a_{116} = CC6B8A74	a_{117} = EA91AC75	a_{118} = 09B7CD76	a_{119} = 27DCEF77

a_{120} = 45021178 a_{121} = 63283279 a_{122} = 824E547A a_{123} = A074767B
a_{124} = BE99977C a_{125} = DCBFB97D a_{126} = FBE5DB7E a_{127} = 190BFC7F
a_{128} = 37301E80 a_{129} = 56564081 a_{130} = 747C6182 a_{131} = 92A28383
a_{132} = B0C8A584 a_{133} = CFEDC685 a_{134} = ED13E886 a_{135} = 0B390A87
a_{136} = 2A5F2B88 a_{137} = 48844D89 a_{138} = 66AA6F8A a_{139} = 84D0908B
a_{140} = A3F6B28C a_{141} = C11CD48D a_{142} = DF41F58E a_{143} = FD67178F
a_{144} = 1C8D3990 a_{145} = 3AB35A91 a_{146} = 58D87C92 a_{147} = 77FE9E93
a_{148} = 9524BF94 a_{149} = B34AE195 a_{150} = D1700396 a_{151} = F0952497
a_{152} = 0EBB4698 a_{153} = 2CE16899 a_{154} = 4A07899A a_{155} = 692CAB9B
a_{156} = 8752CD9C a_{157} = A578EE9D a_{158} = C49E109E a_{159} = E2C4329F
a_{160} = 00E953A0 a_{161} = 1E0F75A1 a_{162} = 3D3597A2 a_{163} = 5B5BB8A3
a_{164} = 7980DAA4 a_{165} = 97A6FCA5 a_{166} = B6CC1DA6 a_{167} = D4F23FA7
a_{168} = F21861A8 a_{169} = 113D83A9 a_{170} = 2F63A4AA a_{171} = 4D89C6AB
a_{172} = 6BAFE8AC a_{173} = 8AD409AD a_{174} = A8FA2BAE a_{175} = C6204DAF
a_{176} = E5466EB0 a_{177} = 036C90B1 a_{178} = 2191B2B2 a_{179} = 3FB7D3B3
a_{180} = 5EDDF5B4 a_{181} = 7C0317B5 a_{182} = 9A2838B6 a_{183} = B84E5AB7
a_{184} = D7747CB8 a_{185} = F59A9DB9 a_{186} = 13C0BFBA a_{187} = 32E5E1BB
a_{188} = 500B02BC a_{189} = 6E3124BD a_{190} = 8C5746BE a_{191} = AB7C67BF
a_{192} = C9A289C0 a_{193} = E7C8ABC1 a_{194} = 05EECCC2 a_{195} = 2414EEC3
a_{196} = 423910C4 a_{197} = 605F31C5 a_{198} = 7F8553C6 a_{199} = 9DAB75C7
a_{200} = BBD096C8 a_{201} = D9F6B8C9 a_{202} = F81CDACA a_{203} = 1642FBCB
a_{204} = 34681DCC a_{205} = 528D3FCD a_{206} = 71B360CE a_{207} = 8FD982CF
a_{208} = ADFFA4D0 a_{209} = CC24C5D1 a_{210} = EA4AE7D2 a_{211} = 087009D3
a_{212} = 26962AD4 a_{213} = 45BC4CD5 a_{214} = 63E16ED6 a_{215} = 81078FD7
a_{216} = 9F2DB1D8 a_{217} = BE53D3D9 a_{218} = DC78F4DA a_{219} = FA9E16DB
a_{220} = 19C438DC a_{221} = 37EA5ADD a_{222} = 55107BDE a_{223} = 73359DDF
a_{224} = 925BBFE0 a_{225} = B081E0E1 a_{226} = CEA702E2 a_{227} = EDCC24E3
a_{228} = 0BF245E4 a_{229} = 291867E5 a_{230} = 473E89E6 a_{231} = 6664AAE7
a_{232} = 8489CCE8 a_{233} = A2AFEEE9 a_{234} = C0D50FEA a_{235} = DFFB31EB
a_{236} = FD2053EC a_{237} = 1B4674ED a_{238} = 3A6C96EE a_{239} = 5892B8EF
a_{240} = 76B8D9F0 a_{241} = 94DDFBF1 a_{242} = B3031DF2 a_{243} = D1293EF3
a_{244} = EF4F60F4 a_{245} = 0D7482F5 a_{246} = 2C9AA3F6 a_{247} = 4AC0C5F7
a_{248} = 68E6E7F8 a_{249} = 870C08F9 a_{250} = A5312AFA a_{251} = C3574CFB
a_{252} = E17D6DFC a_{253} = 00A38FFD a_{254} = 1EC8B1FE a_{255} = 3CEED2FF